Universidad Autónoma de Madrid

Escuela Politécnica Superior

Departamento de Ingeniería Informática

Desarrollo de un widget para presentación de lineas de tiempo y datos multimedia asociados

Luis Panadero Guardeño

Tutor : Alvaro Ortigosa

Trabajo Fin de Grado

Grado en Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

06/07/2014

Resumen

Actualmente hay una gran facilidad de herramientas, métodos y librerías que permiten realizar *casi* cualquier tipo de representación de datos en la web. Sin embargo hay un tipo especial de gráficas, los cronogramas o TimeLines, ya sea por que no se les ha dado importancia o por que cuando surge la necesidad se tiende a recurrir a soluciones propias hechas ex proceso para ello, para las cuales no hay casi librerías que permitan realizarlas con facilidad y con una interfaz adaptable por el desarrollador. Además, hoy en día es prácticamente obligatorio tener en cuenta las interfaces táctiles ya que los móviles y las tabletas se han vuelto ubicuas y usan este tipo de interfaces, además, las pocas librerías pre-existentes no están diseñadas con esta faceta en mente.

El proyecto surge de la necesidad de cubrir los requisitos expresados por el cliente, de presentar un *timeline* que se pueda adaptar fácilmente, que funcione adecuadamente con las interfaces táctiles, además de otros requisitos que las librerías pre-existentes no cumplen de la forma requerida por el cliente.

En este proyecto, se ha intentado abordar el problema de una forma mas general, que permite ser usado en otros casos de uso no previstos inicialmente por el cliente, y además con una mentalidad de código libre (aunque no lo es todavía) de la librería para que el resto de la comunidad pueda beneficiarse de ella y así mismo pueda contribuir en ella y mejorarla.

En el documento que se expone a continuación recoge un pequeño análisis del tipo de gráfica que hablamos, los *timeline*, con algún ejemplo gráfico. También, en estado del arte se comenta las librerías pre-existentes, indicando sus problemas y logros. En entorno de desarrollo, se explica el entorno de desarrollo y tecnologías usadas para su implementación. En el capitulo de Diseño y desarrollo se detalla el diseño y el desarrollo haciendo incapie en algunos detalles de implementación que el autor ha considerado importantes. Y finalmente, en las secciones de pruebas y conclusiones, se evalua lo que se ha conseguido y las posibles mejoras futuras que podría aplicarse.

Palabras clave

Widget, timeline, javascript, cronograma, web, html

Índice

Índice de ilustraciones

Índice de bloques de código

1. Introducción

A medida que se ha extendido el uso de Internet, y en especial la web, han aparecido diversos métodos de presentar la información, por ejemplo, librerías JavaScript tales como **Google Charts**, que permiten crear gráficas a partir de un simple array JavaScript. Sin embargo, hay un caso particular de gráficas para presentación de datos en el que no ha habido casi desarrollo, y sin embargo son muy útiles para mostrar una evolución de eventos históricos o cronológicos.

Este tipo de gráficas, conocidos como "Time Lines", lineas de tiempo o cronogramas, presentan un eje que se corresponde con el tiempo, y presentan eventos puntuales o que ocurren a lo largo de un periodo de tiempo, colocando los alineados con el momento de inicio de dicho evento en el eje (véase la figura 1). Es decir, son gráficas donde se trabaja con un valor que se representa en una sola dimensión.

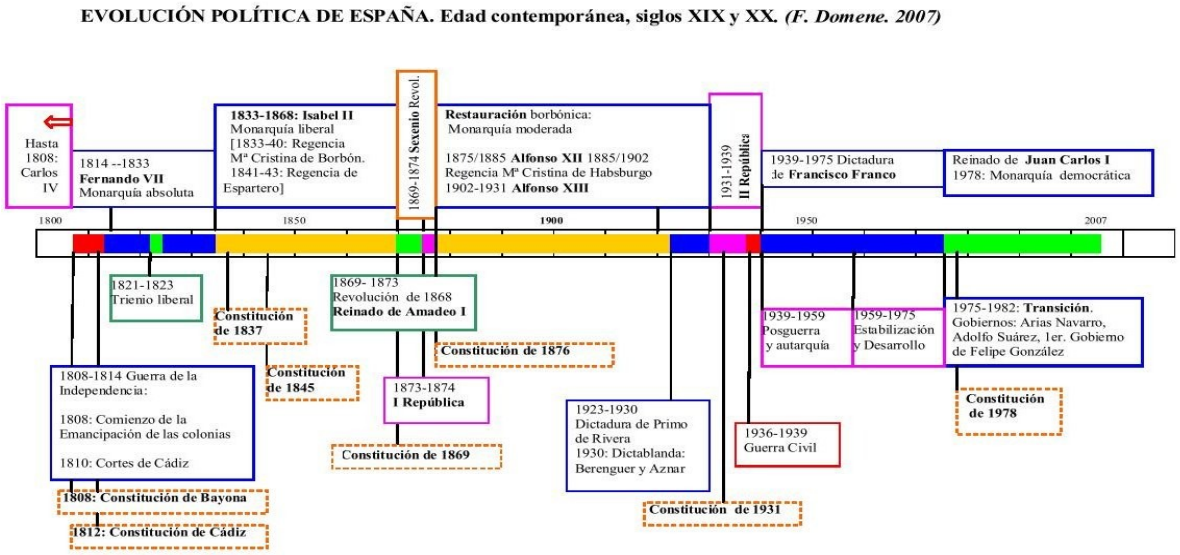

Figura 1: Ejemplo de Linea de Tiempo [1]

1.1 Motivación

Para crear este tipo de gráficas, apenas hay librerías JavaScript u otros métodos reusables para crearlas, siendo lo más habitual encontrar soluciones *ad-hoc* para casos particulares hechas con mapas de imágenes, flash o tablas con imágenes y enlaces.

Con el tiempo, en los últimos años han aparecido ciertas librerías JavaScript que tratan de atacar

este problema de forma más genérica, aunque así presentan sus propias limitaciones y problemas. Este trabajo se centra en el desarrollo de un *widget*, para resolver la necesidad practica del cliente, Digibís S.L., que estas librerías no consiguen solventar de forma totalmente adecuada sus requisitos. Dicha necesidad es la presentación de una linea de tiempo de varios "Polígrafos" [2] , en la que además del cronograma, se presente información detallada de cada autor y su lugar de procedencia en un mapa. Durante el desarrollo del prototipo, se vio que el concepto del *widget* era adecuado para otros casos de uso, con ciertas modificaciones, lo cual impulso mas su desarrollo con un diseño genérico, abarcando de esta forma una mayor cantidad de casos de uso y necesidades.

1.2 Objetivos

Los principales objetivos indicados por el cliente, que se quieren alcanzar en el desarrollo de este *widget* son :

- Presentación de una cantidad "grande" (≥90 en el caso de los Polígrafos) de entradas en el cronograma.
- Cada entrada, u evento temporal, muestra una etiqueta emplazada según la fecha/hora del evento.
- Origen de los datos, mediante un array de "objetos" JavaScript, lo más sencilla posible.
- Sistema de eventos que se disparen ante las interacciones del usuario.
- Poder "desplazarse en el tiempo" y realizar "zoom" para poder observar eventos con mayor detalle.
- La interfaz sea rápida y sencilla de usar.
- *Widget wraper* que permita ser usado como solución *ad-hoc* para ciertos casos de uso típicos del cliente.
    - Cada entrada tiene asociado una información multimedia que se desea mostrar cuando se seleccione dicha entrada.
    - Cada entrada puede tener asociada una localización geográfica que se mostrará en un mapa.
    - Posibilidad de navegación de las entradas, mediante su selección en el mapa o mediante botones de adelante y atrás.

1.3 Estructura del trabajo

El presente trabajo se estructura en 6 capítulos, que se detallan a continuación :

- **Capitulo 1: Introducción.** Se define el contexto, motivación y objetivos del presente trabajo. Así mismo, se detalla la estructura del mismo y se especifica la notación o terminología a usar.
- **Capitulo 2: Estado del arte.** Estudio sobre otros *widgets* y librerías que tiene un objetivo o funcionalidad similar, analizando que tienen de similar y de distinto, así como sus ventajas y desventajas.
- **Capitulo 3: Entorno de desarrollo.** Se explica las herramientas y tecnologías usadas en el desarrollo.
- **Capitulo 4: Diseño y Desarrollo.** Se describe la ingeniería software usada en el presente trabajo, los componentes usados y los problemas mas importantes que se han tratado.
- **Capitulo 5: Pruebas.** Descripción breve de las pruebas realizadas para comprobar su correcto funcionamiento, y el desempeño logrado..
- **Capitulo 6: Conclusiones.** El resultado final del actual desarrollo del *widget*, indicando sus capacidades, objetivos deseados logrados y posibles mejoras.

1.4 Notación y terminología.

A continuación se detallan ciertos términos o notaciones que se usan a lo largo del presente trabajo:

TimeLine o timeline	Cronograma donde se presenta una serie de eventos cronológicos usando un eje asociado al tiempo.
Entrada	Cada evento del *timeline* con su información asociada.
Dato multimedia	Texto, imágenes, videos, localización geográfica (mapa), asociado a una entrada.
Bandera	Cada entrada en el *timeline* tiene una caja u *bandera*, con un breve texto que actuá como etiqueta del evento.
Momento inicial	Momento del tiempo en la que ocurre o empieza un evento u entrada del *timeline*. Define donde se colocara la bandera de dicha entrada en el *timeline* respecto al eje de tiempo.
Momento final	Momento del tiempo en la que termina un evento. Si no esta presente, se tomara el evento como un evento puntual en el tiempo.
Ventana de tiempo	Rango de tiempo actualmente visible en el *widget*. Controlable por parte del usuario.

Cursor	Momento del tiempo en que el usuaria tiene centrado la ventana de tiempo.
Bandera seleccionada	Evento del TimeLine que el usuario tiene actualmente seleccionado.
Zoom	Cambiar el intervalo de tiempo que cubre la ventana de tiempo.

2. Estado del arte

El desarrollo de este *widget* responde ante ciertas limitaciones encontradas en librerías ya existentes que tratan de atacar el problema. Además, durante el desarrollo de la misma, apareció una nueva librería que también ataca el mismo problema, pero con un enfoque más a la edición de los datos por parte del usuario en vez de la simple presentación. A continuación procederemos a ver que diferencias y problemas presentan dichas librerías, y que el *widget* de este trabajo trata de solventar.

2.1 timeline.js y timemapper

Esta librería [3] para la elaboración de lineas de tiempo, fue la primera que se probó y ha sido la principal motivo de la creación de este trabajo.

Presenta una interfaz gráfica sencilla y fácil de usar, además de funcionar con suavidad y tener cierto soporte táctil. Su enfoque esta centrado en presentar una serie de eventos que no se solapan mucho en el tiempo, y con cada evento mostrar una imagen, texto, mapa o vídeo asociado. Permite hacer “zoom” con el que se puede agrandar o reducir la ventana de tiempo visible, aparte de poder arrastrar el *timeline* para cambiar el intervalo de tiempo visible sin necesidad de seleccionar otra entrada.

Figura 2: Ejemplo de timeline.js

Sin embargo presenta una serie de problemas que lo hacen un tanto frustrante de usar en cuanto se sale del caso de uso pensado por el creador. En primer lugar, esta diseñado para que cada evento tenga un dato multimedia OBLIGATORIO que se presente con el *timeline*. Si, por ejemplo, solo se quisiera presentar el *timeline* con el texto breve y sus fechas, el resultado es que se pierde espacio en la página con información redundante. Dicho de otra forma, el carrusel y el *timeline* son inseparables.

En segundo lugar, el layout del *timeline* con el dato nultimedia que se presenta, está definido por la propia librería y resulta complicado de modificar. Así mismo, el estilo del *widget* está en gran medida embebido en el propio código JavaScript. A esto se le añade que no está bien documentado donde están o cuáles son esos parámetros a modificar para alterar el aspecto o el layout. Además, dentro del código JavaScript, se ha incluido funciones de uso muy puntual y que aumentan el tamaño de la librería (¡como toda una implementación de AES en JavaScript!). Alguna de estas limitaciones de estilo, se pueden solventar parcialmente, embebiendo HTML dentro de los datos, pero resulta una solución un tanto compleja de usar por los efectos imprevistos que puedan suceder del CSS propio de **timeline.js**.

En resumen, esta librería funciona bien, si se va a usar con el caso de uso y diseño web que tiene en

mente el creador, hasta el punto que todos los ejemplos listados en su página son prácticamente idénticos, solo variando el contenido. Intentar modificarlo, puede ser verdaderamente farragoso.

La librería **TimeMapper** usa timeline.js y recline.js , además de integrar un mapa de Leaflet y permitir tomar diversas fuente de datos, como una hoja de **Google Drive**, lo cual lo vuelve en muy útil para blogueros y periodistas que quieran usarlos para artículos en la web, pero sigue presentando prácticamente la misma inflexibilidad que el original. La inflexibilidad y la imposibilidad de modificar el layout resultante de una forma razonable, fue la principal razón de la creación de este proyecto.

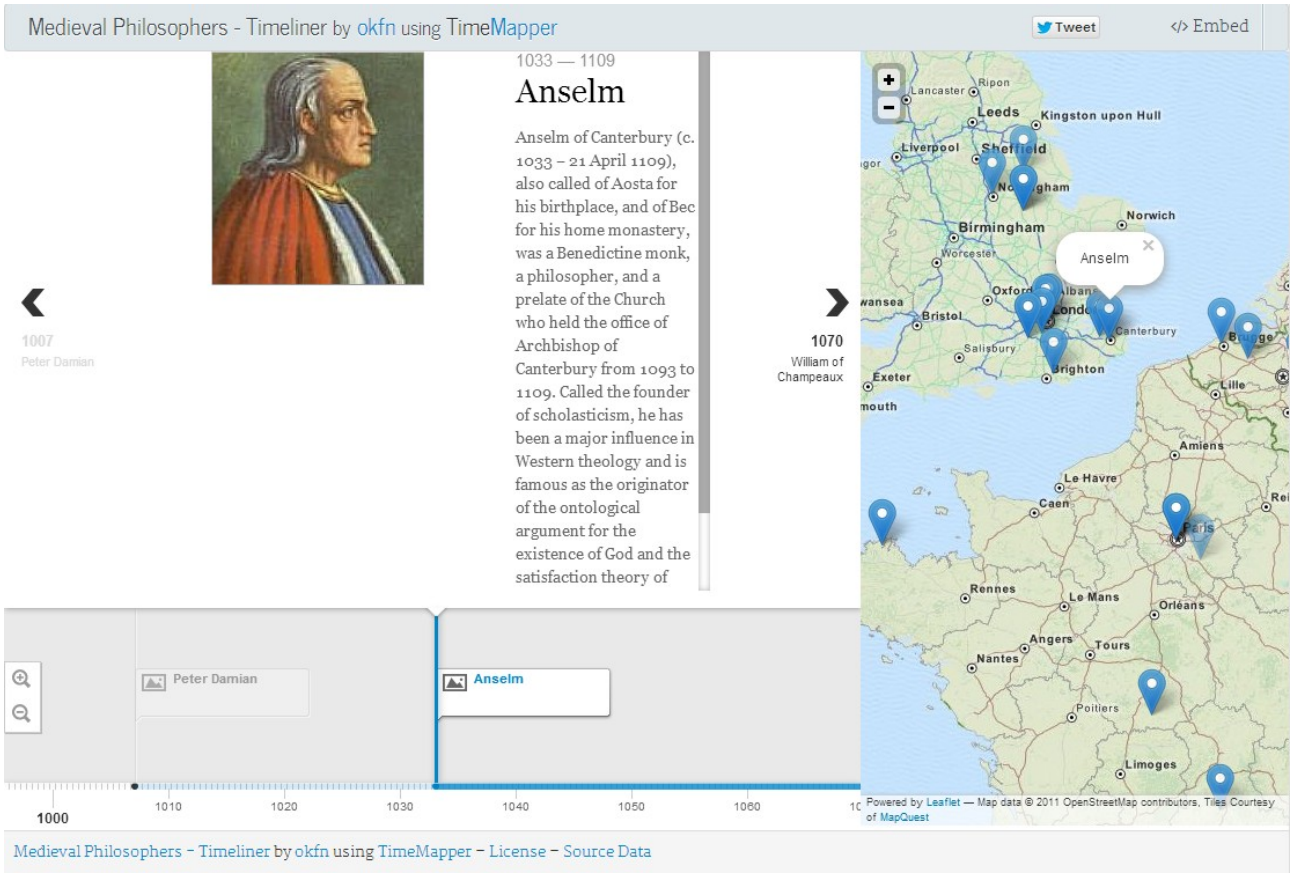

Figura 3: Ejemplo de TimeMapper

2.2 SIMILE timeline

SIMILE timeline [4], se trata de un *widget* para la presentación de *timelines* dentro del proyecto conjunto SIMILE del M.I.T. y C.S.A.I.L. Posee una interfaz bastante agradable y permite presentar con rapidez una gran densidad de entradas. Sin embargo tiene dos problemas particulares. En primer lugar, el proyecto esta abandonado, lo cual ha hecho que se haya quedado estancado en las posibilidades de los navegadores del 2009, y que además, haya alguna funcionalidad que ya no funciona actualmente como debiera. El otro inconveniente es que está diseñado pensando mas en la

presentación en si del *timeline* que en otra cosa. Esto conlleva que por ejemplo tenga funcionalidades interesantes como filtrar entradas, múltiples minimapas (capas), anotaciones, etc pero sea bastante limitante si se quiere mostrar algo más que un breve texto o una pequeña imagen a cada entrada. Además, no tiene capacidad de hacer "zoom", es decir, controlar el tamaño de la ventana de tiempo visible.

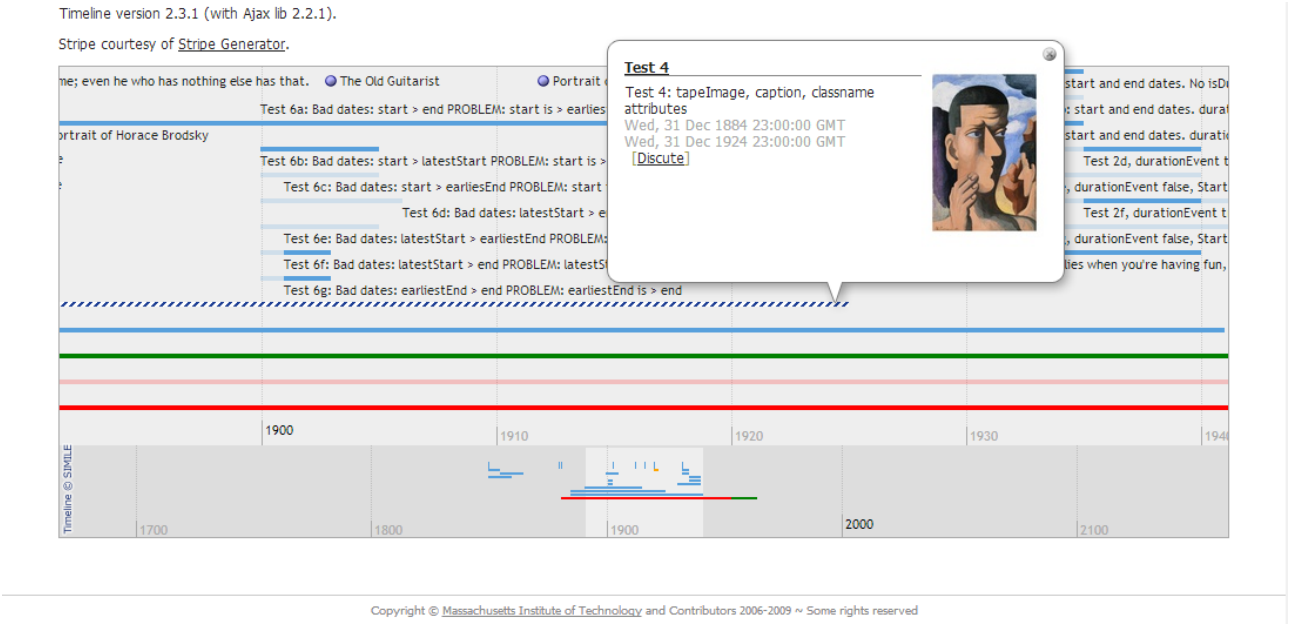

Figura 4: Ejemplo de SIMILE timeline

2.3 vis.js

vis.js [5] es una librería muy reciente (apareció sobre el 15 de Abril en Hacker News [6]), y quizás la que realmente se acerca a lo deseado por el cliente, e incluso la supera en algunos detalles como tener capacidad de poder editar los datos por parte del usuario. También tiene otras funciones extra, como la representación de grafos.

Podemos destacar dos facetas en las que la implementación de este trabajo supera a vis.js. En primer lugar, en el trabajo se ofrece una solución que permite tanto presentar un *timeline* con o sin datos multimedia asociados, **vis.js** necesita que se le añada código u otros widgets para presentar algo más que el *timeline* puro. En segundo lugar, al tener la funcionalidad extra para editar los datos, hace que la librería sea más pesada (600KiB la versión minimizada). En nuestro caso, se opto por un planteamiento mas modular que permite "desprenderse" del *widget wrapper*.

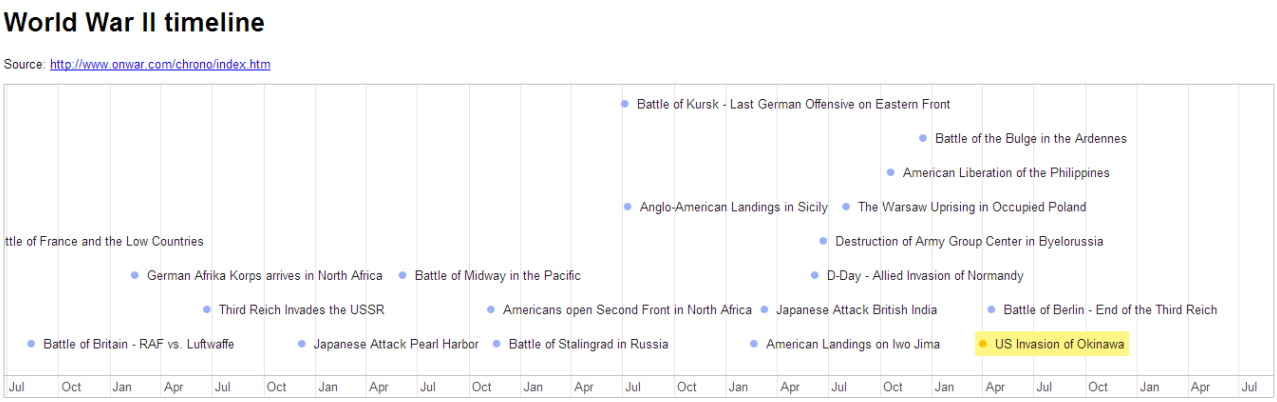

Figura 5: Ejemplo de vis.js

Su funcionamiento se podría resumen en que altera el árbol DOM dinámicamente, creando y destruyendo nodos según entra o salen las banderas de cada evento de la ventana de tiempo, y reciclándolos. Es decir, tiene un funcionamiento similar a los *widgets* de presentación de mapas. Esto le permite lidiar con una gran cantidad de eventos, siempre y cuando se auto-limite a una cantidad razonable la cantidad de banderas visibles a la vez, y este sistema de funcionamiento se esta contemplado para futuras mejoras del *widget* resultante de este trabajo.

3. Entorno de desarrollo

En el desarrollo de este trabajo, se ha utilizado ciertas tecnologías ya existentes para facilitar y acelerar el desarrollo. Así mismo fue necesario establecer un entorno de desarrollo que permitiese facilitar y agilizar el desarrollo de un *widget* JavaScript. Por lo tanto, se ha usado con éxito diferentes tecnologías y técnicas. Algunas de ellas están bien probadas, como GIT, y otras son de reciente aparición, tal como el pre-procesador LESS.

3.1 GIT y Bitbucket

Para llevar un control del código fuente, permitir desarrollarlo en distintos sitios y poder llevar desarrollo en paralelo de distintas funcionalidades o mejoras, se decidió usar un sistema de control de versión del código fuente.

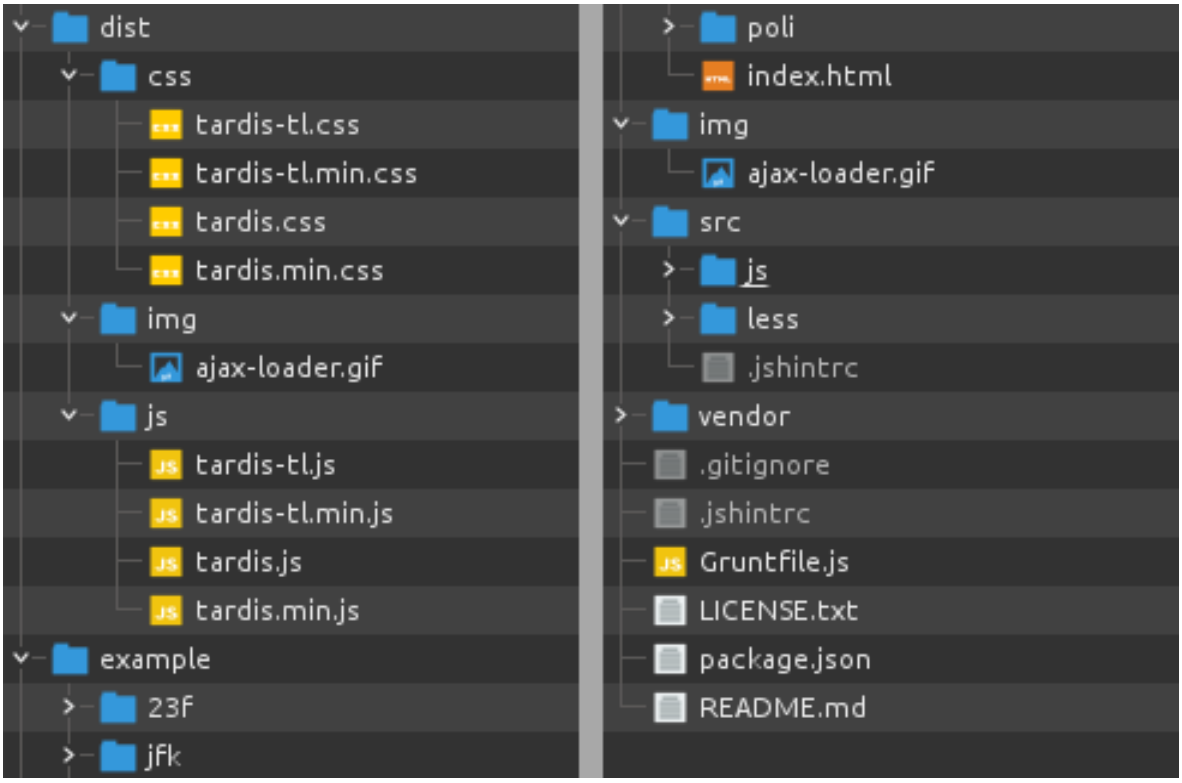

Figura 6: Estructura de directorios y ficheros

Particularmente, se optó por usar el sistema de control de versiones distribuido GIT, aparte de que es tremendamente popular y está muy bien documentado su funcionamiento. Al ser distribuido, permite tener alojado un repositorio en un servicio web gratuito, y al mismo tiempo que permite trabajar con copias locales en distintas estaciones de trabajo.

La estructura de directorios en el repositorio, es la que se puede ver en la figura 6. En **/src/** se almacenan los ficheros fuente de JavaScript y LESS. En **/img/** se almacenan las imágenes que se usan en el proyecto. En **/dist/** están los ficheros generados y preparados para la distribución. En **/example/** están las demos con sus sets de datos y su propio fichero LESS. En **/vendor/** se guardan copias de las librerías que el proyecto tiene dependencia y/o son necesarias para los ejemplos. Finalmente, en el raíz esta el fichero de licencia, los ficheros necesarios para Grunt, npm, jshlint y el fichero README.

El flujo de uso de GIT se reparte en varias ramas, siendo la rama "**master**", la rama con el código estable y que solo se actualiza para arreglar algún fallo detectado o para un cambio de versión. Aparte esta la rama "**develop**" que es donde se centra el desarrollo de las nuevas versiones del código fuente. Puntualmente se puede crear nuevas ramas para solucionar fallos o desarrollar alguna nueva característica, que luego se reunificaran (*merge*) en la rama "**master**" si son soluciones de fallos, y/o en la rama "**develop**" si se trata de nuevas características. Por lo tanto, a groso modo, se sigue el esquema de ramas que se usa en casi todos los proyectos con GIT y que han hecho tan

popular este sistema de versiones. Como nota interesante, la rama "**develop**" mantiene vació el contenido del directorio **/dist/**, así que cada cambio que se realice, no fuerza a estar aplicando cambios en GIT de los ficheros generados, sin embargo esto requiere que cuando se realiza un *merge* en la rama "**master**", haya que volver a generar los ficheros antes de realizar el *commit*.

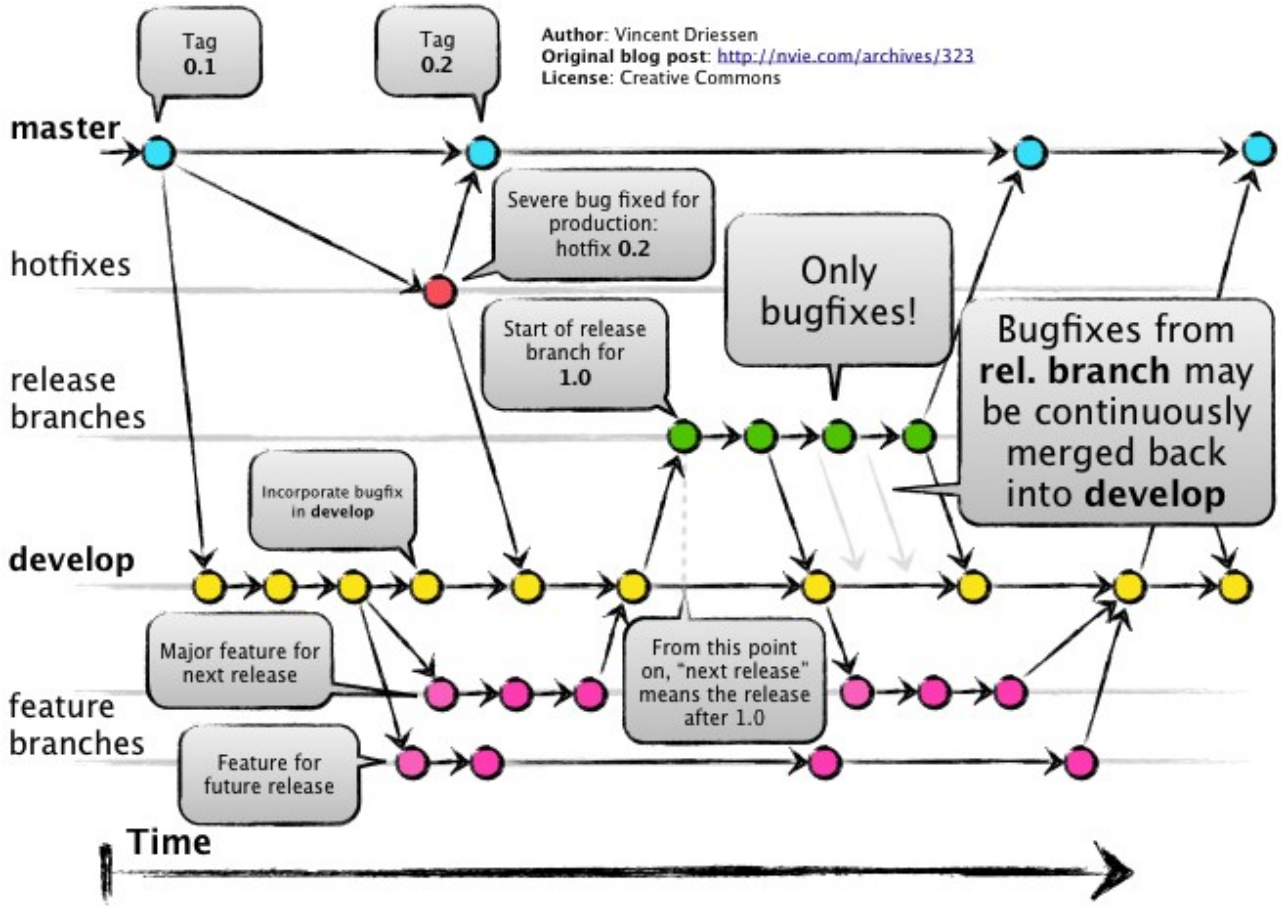

Figura 7: Flujo típico de proyectos que usan GIT

Para el alojamiento del "repositorio central" se opto por usar el servicio gratuito Bitbucket, ya que permite crear un repositorio GIT privado, llevar control de incidencias y tener una *wiki* para alojar la documentación del funcionamiento de los *widgets*.

3.2 LESS – Pre-procesador de CSS

Las hojas de estilo en cascada o **CSS**, permiten definir, en teoría, el aspecto de forma independiente al contenido y layout de una página web. Sin embargo, su sintaxis a veces puede provocar efectos secundarios inesperados, y tiene carencias notables como carecer de variables o macros. Ante la creciente complejidad y repetición para lograr tener una cierta coherencia entre navegadores, han hecho acto de presencia "pre-procesadores" de hojas de estilo. Particularmente se han hecho populares el pre-procesador **LESS** y el pre-procesador **SASS**. Ambos tiene características muy similares, aunque actualmente se considera más potente el pre-procesador **SASS**.

Para el desarrollo de este trabajo, se decidió usar **LESS** ya que el cliente viene usándolo para sus propios desarrollos y el autor ha adquirido experiencia con el mismo durante las primeras fases del desarrollo del prototipo.

Las ventajas que aporta usar **LESS** sobre **CSS** puro son :

- Evita repeticiones innecesarias, al poder usar macros y metra-programación de los selectores.
- Uso de variables para alojar de forma localizada la paleta de colores, tamaño de las fuentes, bordes, etc. Por ejemplo, hace innecesario recorrer todo el fichero si solo se quiere modificar la paleta de colores. También permite cambiar el esquema de colores, tamaños, etc... simplemente añadiendo un nuevo fichero LESS que remplace los valores de dichas variables.
- Añade funciones al lenguaje, tales como funciones matemáticas, manipulación de colores, manipulación de cadenas de texto y listas.
- Sintaxis jerárquica obvia.

Sin embargo **LESS** requiere compilar los ficheros LESS para generar el **CSS** final. El compilador oficial de **LESS** esta elaborado en JavaScript, y se puede usar independientemente con **Node.js** o siendo llamado desde un navegador web para que procese el fichero LESS en la carga de la misma. Esta ultima opción solo se recomienda durante el desarrollo, ya que ralentiza la carga de la página notoriamente.

3.3 Node.js y npm

Con el objetivo de agilizar el proceso de desarrollo, se opto por usar el compilador de **LESS** desde **Node.js**. **Node.js** es un maquina virtual de JavaScript que funciona independientemente de un navegador, y que ofrece un entorno para desarrollar aplicaciones y servicios basados en JavaScript y que usen I/O asíncrona. También se puede usar como parte de un entorno de desarrollo para JavaScript, el cual es este caso.

Npm es un gestor de paquetes para **Node.js**, que permite, usando un fichero **JSON**, `package.json` , indicar dependencias de un proyecto de **Node.js**. En este trabajo, se usa para auto descargar e instalar las herramientas del entorno de desarrollo, tales como grunt, el compilador de LESS, jshlint, etc... También se podría usar para auto-descargar las dependencias de la librería, pero hemos creído que no era adecuado, al ser el trabajo una librería JavaScript para un navegador web y

no para **Node.js**.

```
{
  "name": "Tardis.js",
  "version": "0.2.0",
  "description": "Toolset of widgets to display interactive timelines with multimedia
and geographic data associated.",
  "author": {
    "name": "Luis Panadero Guardeño",
    "email": "luis.panadero@gmail.com"
  },
  "engines": {
    "node": "^0.8.0"
  },
  "devDependencies": {
    "grunt": "^0.4.3",
    "grunt-contrib-jshint": "^0.6.0",
    "grunt-contrib-concat": "^0.3.0",
    "grunt-contrib-uglify": "^0.2.0",
    "grunt-contrib-watch": "^0.4.0",
    "grunt-contrib-clean": "^0.4.0",
    "grunt-contrib-less": "~0.8.3",
    "grunt-contrib-copy": "^0.5.0"
  },
  "bugs": "https://bitbucket.org/Zardoz84/tardis/issues",
  "licenses": [
    {
      "url":
"https://bitbucket.org/Zardoz84/tardis/raw/e2db4782e4f7c8136e60ea5b6c57ca19d1c80ae0/L
ICENSE.txt"
    }
  ],
  "private": true
}
```

Código 1: package.json

Como se puede apreciar en Código 1, la sintaxis es sencilla y fácil de entender. Hay una serie de entradas con información sobre el nombre del proyecto, versión del paquete, versión del motor JavaScript a usar, una lista de *scripts* opcionales a ejecutar, y lo más importante para nosotros, una lista de dependencias con indicadores de la versión a usar.

La lista de dependencias esta definida por un array asociativo donde la clave es el paquete, y el contenido de cada entrada, es una identificador de la versión que se requiere. El identificador de versión puede requerir una versión igual o superior si es precedido por el símbolo “^” , o se puede requerir una versión aproximada con el símbolo “~”.

En nuestro caso, se declara que depende de **grunt**, así como ciertas sub-herramientas y lanzadores que se usaran con grunt, tales como *grunt-concat*, *grunt-uglify*, *grunt-watch*, *grunt-copy*, y sobretodo del compilador de **LESS** y el *linter* **jhsint**. Particularmente, se requiere la versión 0.8.3 del compilador **LESS**, ya que es la versión que usa el cliente en su entorno de desarrollo.

Con el fichero `package.json`, solo basta con ejecutar `sudo npm install -g grunt-cli` y `npm install` para que se auto-instalen las herramientas del entorno de desarrollo de forma automática. Dicha instalación se hace de forma local al proyecto, en el sub-directorio "`node_modules`".

3.4 Grunt

Grunt es un equivalente de **Make** para proyectos basados en JavaScript. De forma similar a **Make**, se define un fichero `Gruntfile.js` que actuá de forma similar a un `Makefile`.

A diferencia de **Make**, el fichero `Gruntfile.js` se define una función *wrapper* que es ejecutada por **Grunt**. En dicha función, se realiza llamadas a métodos del objeto `grunt`, con los que se configura y definen tareas o *task* de Grunt. Con `grunt.initConfig()` que toma como parámetro un objeto de configuración, se puede definir parámetros de configuración usados por las tareas. Cada tarea puede tener uno o varios objetivos o *targets*.

```
'use strict';

module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({ ...

    // clean dist files
    clean: {
      js: {
        src: './dist/js'
      },
      css: {
        src: './dist/css'
      },
      img: {
        src: './dist/img'
      }
    },

    ...

  });
```

Código 2: Ejemplo de configuración de una tarea con grunt

Como se puede ver en el ejemplo de cuadro Código 2, se define que para la tarea *clean*, hay los targets *clean:js*, *clean:css* y *clean:img*, que apuntan a los directorios **./dist/js**, **./dist/css** y **./dist/img**.

Para definir tareas, se puede usar dos métodos. El primero, es cargando una tarea dependiente de algún modulo de npm, como por ejemplo con "`grunt.loadNpmTasks('grunt-contrib-clean');`" donde se registra una tarea llamada *clean* que, usando la configuración bajo la entrada *clean* del objeto de configuración, limpiara los ficheros del proyecto. En el ejemplo anterior, limpiara los directorios **./dist/js**, **./dist/css** y **./dist/img**.

El segundo método de definir tareas, es con "`grunt.registerTask(...)`", que nos permite agrupar una serie de tareas dentro de una, y que se ejecutaran en el mismo orden en que se listan.

```
// Tarea de compilar para distribuir
grunt.registerTask('dist', [
  'less:dist',
  'copy:imgs',
  'concat:dist_tardisjs',
  'concat:dist_tardis_tl',
  'jshint'
]);
```

Código 3: Ejemplo de tarea que agrupa a otras

El ejemplo mostrado en el cuadro Código 3, crea una tarea llamada "dist" para generar los ficheros finales de distribución, que ejecuta la tarea de lanzar el *linter* **jshint**, el compilador **LESS** con el target *dist*, la tarea de copiar las imágenes necesarias (*copy* con el target *imgs*), las tareas de concatenar los ficheros JavaScript fuente para generar los ficheros JavaScript finales (*concat* con el target *dist_tardisjs* y con el target *dist_tardis_tl*) y finalmente el *linter* para que revise el código concatenado.

Con este ultimo método, se puede definir la tarea especial "default" que sera ejecutada cuando se llama a **Grunt** sin ningún parámetro (el equivalente a la entrada "all:" en un makefile).

En nuestro caso, se han definido tres tareas agrupadas con gran importancia :

- **dist** : Se encarga de generar los ficheros de distribución sin pasar por *uglify*, e indicando al compilador **LESS** que no compacte el CSS final. Estos ficheros serán : `tardis-tl.js`, `tardis.js`, `tardis-tl.css`, `tardis.css` y `ajax-loader.gif`
- **dist-min**: Análogo a la tarea **dist**, pero pasa los ficheros por *uglify* para generar un fichero JavaScript más compacto e indica al compilador **LESS** que trate de compactar el fichero CSS final. En este caso serán : `tardis-tl.min.js`, `tardis.min.js`, `tardis-tl.min.css`, `tardis.min.css` y `ajax-loader.gif`
- **demo**: Lanza **dist** y **dist-min**, aparte de generar los ficheros CSS que usa los ejemplos incluidos. Además, indica al compilador LESS que genere información de *debug* para saber la procedencia de una regla CSS en las herramientas de desarrollo web de los navegadores Chrome y Firefox.

Hay una serie de tareas importantes, que se pueden usar solas, aparte del uso que se da dentro de dist, dist-min y demo :

- **clean** : Análogo a un `make clean`. Borra los ficheros temporales y los ficheros de distribución.
- **jshint** : Lanza el *linter* de JavaScript.

Hay una tarea especial llamada **watch**, que hace que **Grunt** se quede vigilando una serie de ficheros (por ejemplo los ficheros fuente de JavaScript y LESS), y que cuando detecta un cambio en ellos, lanza una o varias tareas. Con ello, se puede conseguir que las modificaciones en el código, sean automáticamente procesadas, y se actualicen los ficheros de los ejemplos y de distribución. Por lo tanto, solo requiriendo una simple recarga del navegador para ver el efecto de los cambios en el código fuente.

Otra tarea importante es **concat**, que permite concatenar varios ficheros fuente en uno solo. En nuestro caso, se usa para generar los ficheros de distribución `tardis.js` y `tardis-tl.js` a partir de `/src/js/tbegin.js` ,`/src/js/tardis-browser.js` ,`/src/js/tardis-trace.js` ,`/src/js/tardis-tl.js` ,`/src/js/tardis-wrap.js` y `/src/js/tend.js`. También se usa para añadir un *banner* a los ficheros JavaScript generados con la información de versión y licencia.

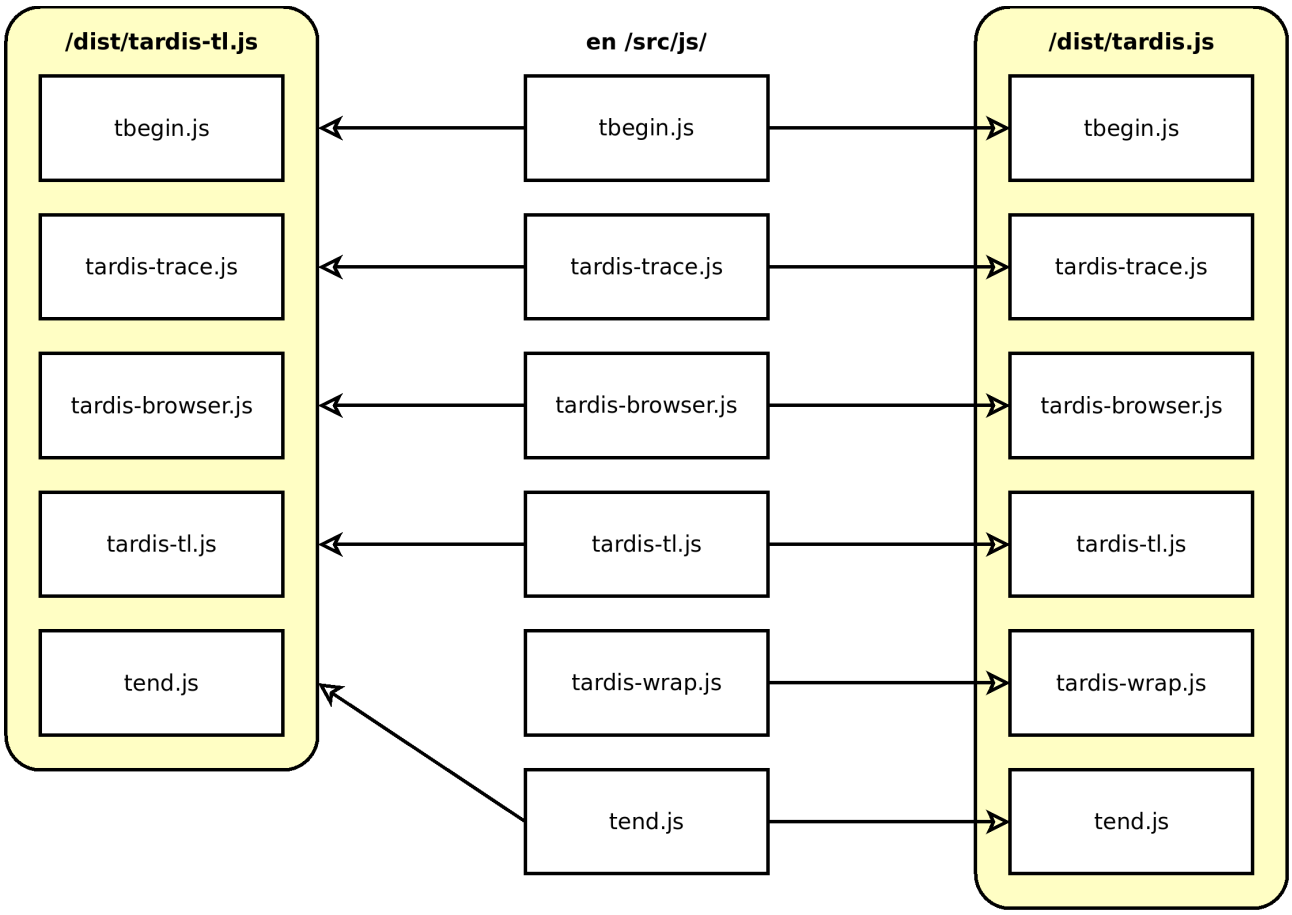

Figura 8: Esquema de concatenación de los ficheros fuente

Como se observa en el gráfico, la diferencia entre ambos ficheros generados básicamente es la inclusión de `tardis-wraper.js` . `tbegin.js` y `tend.js` se usan para encerar el código generado en una función (ver sección jQuery). `tardis-browser.js` contiene código que abstrae del navegador, aparte de informar al *widget* de que navegador se esta usando para ciertos casos especiales. `tardis-trace.js` contiene la función para generar las trazas de funcionamiento del *widget*. Normalmente esta desactivado, y para activar las trazas hay que poner la variable `tardis_debug_trace` a `true` . `tardis-tl.js` es el código del *widget* en si, y `tardis-wraper.js`, es el código del *wraper* del *widget* con el carrusel y el mapa.

3.5 Linter jshlint

Jshlint se trata de un *linter* escrito en JavaScript, que parsea ficheros generados JavaScript, siguiendo una configuración definida en el fichero `.jshintrc`, aunque se puede definir otro fichero distinto con la configuración de la tarea en **Grunt**. En nuestro caso, se tiene dos ficheros, uno para el código JavaScript y otro para los ficheros de **Grunt** y **npm**.

Con el uso de un *linter*, se permite garantizar un cierto nivel de calidad del código JavaScript, prohibiendo ciertas practicas no consideradas seguras o recomendables, y forzando el uso de practicas recomendadas como el "modo estricto" de JavaScript. Si el código incumple dichas normas, **jshint** generara un aviso, y abortara la tarea de Grunt en curso, informando que lineas han incumplido alguna norma para que el desarrollador pueda tomar medidas al respecto.

Las principales reglas que se han establecido en el *linter* para procesar el código fuente JavaScript, obligan a usar el modo estricto de JavaScript, a evitar múltiples definiciones de una misma variable, a usar las comparaciones estrictas (**===** y **!==**), no declarar variables que no se usen, el uso de { y } en los bloques uni-linea para bucles y condicionales, definir las variables antes de usarlas, etc. También se activa una serie de opciones para indicar que usamos **jQuery** en un navegador web, y la pre-existencia del objeto L usado por **Leaflet**.

4. Diseño y desarrollo

Durante el desarrollo, se ha pasado por diferentes fases, elaborándose inicialmente un prototipo que resolvía el problema planteado para una situación particular (*TimeLine* de polígrafos de la biblioteca digital de la fundación Ignacio Larramendi), que luego se fue desarrollando para un uso más genérico. Durante dicha evolución, se abandono el uso de **Bootstrap**, se modularizo el código JavaScript en partes diferenciadas, y se resolvió algunos problemas de presentación e interacción que se pudieron observar en el prototipo inicial.

Ante el deseo de que el *widget* fuese flexible y adaptable, pero a la vez permitiese generar una presentación de los datos multimedia asociados a cada entrada de forma rápida y sencilla por parte de otros desarrolladores, se decidió modularizar la librería generada en dos *widget*s : **tardis-tl.js** y **tardis.js** . **tardis-tl.js** es el *widget* principal que representa visualmente el *timeline* y ofrece al desarrollador una serie de eventos, cuando el usuario selecciona una bandera de un evento. Mientras **tardis.js**, integra en un *widget wraper*, la funcionalidad de **tardis-tl.js**, una implementación ligera de *carrusel* y un mapa para facilitar la presentación de datos multimedia y geográficos de los eventos.

4.1 tardis-tl.js

La parte encargada del *widget* de *timeline*, se da a conocer como **tardis-tl.js** , para así poder distinguirla claramente de la agrupación con el *wrapper*.

Su concepción se inspira en los detalles de funcionamiento de **timeline.js**, que esencialmente se

resume en tener dos `<div>` anidados, en donde el exterior, define donde va a estar visible el *widget* visible en la página y tiene con la propiedad `overflow: hidden`. El `<div>` interno tomando el tamaño y desplazamiento adecuados, muestra una ventana de un contenido más grande que el contenedor. Dicho de otra forma, es más grande por dentro que por fuera (de ahí el nombre de Tardis). En el `<div>` interno, se crean las banderas para cada evento siendo cada una posicionada en el eje de tiempo con la propiedad `left` y un valor porcentual. Así que para desplazar la ventana de tiempo, solo hay que desplazar el `<div>` interno respecto al externo, y para realizar "zoom", solo hay que cambiar la dimensiones del `<div>` interno. El funcionamiento es bien sencillo y fácil de implementar, y escala relativamente bien hasta una cantidad de entradas del orden de 100, y con niveles de zoom adecuados para varios sets de datos.

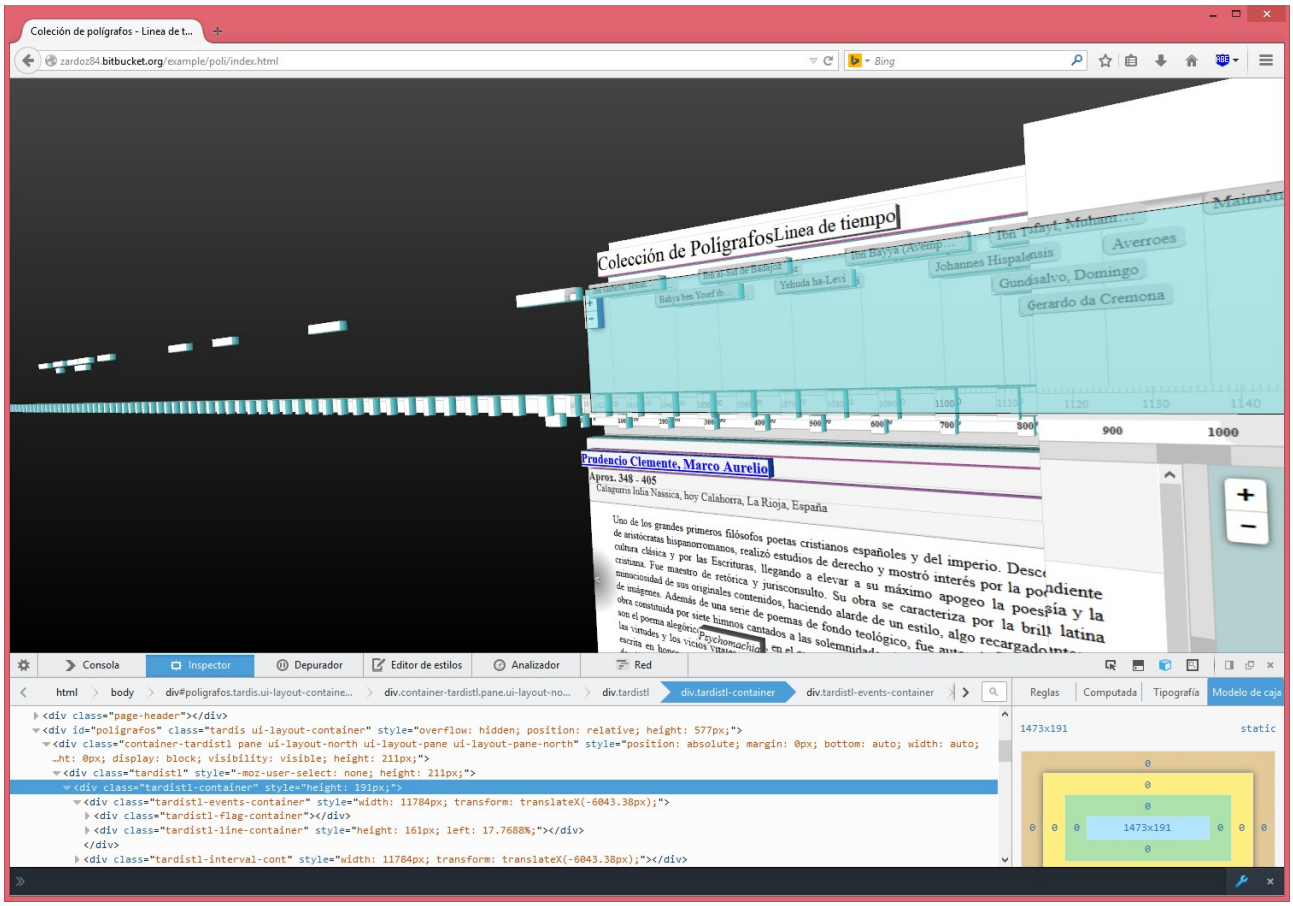

Figura 9: Visualización del efecto "Tardis"

Conceptualmente el diseño general del *widget* consiste en un objeto JavaScript que posee unos métodos públicos y otros "privados". El constructor genera el DOM para presentar el *widget* dentro de un selector de jQuery, y además registra los *handlers* de los eventos, aparte de registrar dos eventos en jQuery, para que código externo al objeto pueda saber cuando el usuario final ha cambiado de elemento seleccionado. Este objeto JavaScript, es envuelto en un plugin de jQuery que

expone solo los métodos públicos, y además permite acceder a ellos en una forma más natural al código jQuery.

4.1.1 Interfaz de Usuario

La interfaz de usuario trata de ser lo más obvia y fácil de usar que se pueda. Además se trata de tener en cuenta las interfaces táctiles a la hora de darle unos ciertas dimensiones adecuadas.

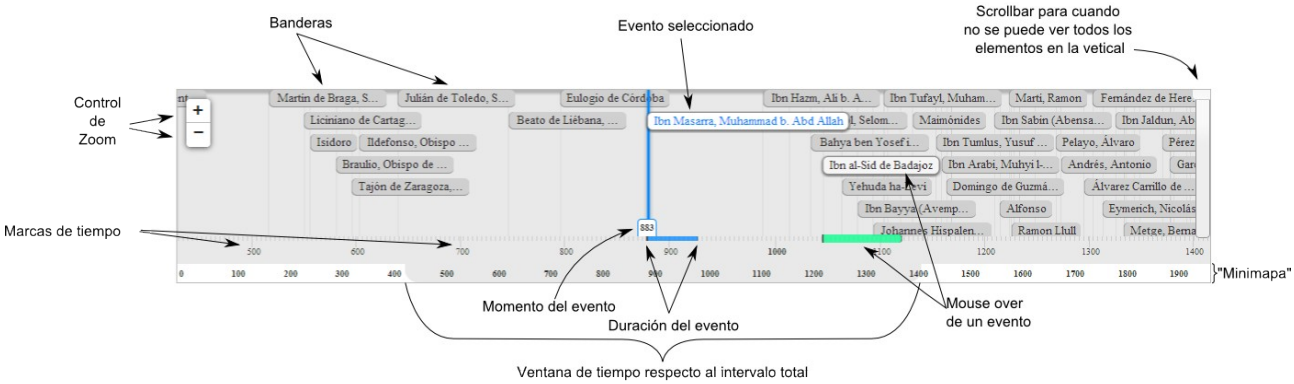

Figura 10: Elementos de la Interfaz de Usuario

Como se puede ver en la figura 10, consiste en varios elementos :

- Banderas de los eventos. Representa cada evento temporal a ser mostrado en el *timeline*. Se colocan horizontalmente en función del momento de inicio del evento temporal, y verticalmente se desplazan para que no se tapen entre si.
- Marcas de tiempo. Son unas marcas orientativas que ayudan al usuario a percibir la escala temporal del *timeline* y la ubicación temporal aproximada de los eventos representados por las banderas.
- "Minimapa". Consiste en una visión del intervalo total de tiempo, que puede tener opcionalmente marcas de tiempo, y donde hay una representación de la ventana de tiempo actualmente visible. Se puede arrastrar dicha representación para mover la ventana de tiempo visible.
- Scrollbar. Ya que dependiendo del nivel de zoom y el set de datos a usar, puede ocurrir que haya muchos eventos cercanos entre ellos y que se apilen para evitar taparse entre si, es necesario dar al usuario un respaldo visual de que hay mas banderas ocultas en la vertical. El usuario puede usar el scrollbar para mover la vertical y ver las banderas que no caben en la vertical.

- Controles de zoom. Se presenta unos controles obvios de zoom para poder agrandar o reducir el tamaño de la ventana de tiempo visible.
- Bandera u evento seleccionado. Se destaca sobre los demás, usando efectos de sombras, y con cambios de color. Además, se muestra el tiempo "exacto" de inicio del evento, y se destaca sobre las marcas de tiempo, su duración.

El usuario puede arrastrar el fondo en que están situados las banderas para mover en horizontal y en vertical. El desplazamiento horizontal, desplaza la ventana de tiempo visible, y el desplazamiento vertical, permite ver banderas que no caben en la vertical.

4.1.2 Unidad y representación del tiempo

Un problema que se plantea cuando se trabaja con fechas y tiempos, es la complejidad de los calendarios (véase por ejemplo el bug de Java 4639407 [7]). En nuestro caso, se opto por tratar de evitar dicha complejidad intentando seguir el principio KISS. En el caso del prototipo inicial, se decidió trabajar con años directamente, ya que en el set de datos, no se usan fechas con mayor precisión. Sin embargo, cuando se decidió dar el paso a algo más genérico, se vio la necesidad de abstraer la representación del tiempo, usando unidades de tiempo abstracto.

La abstracción de la unidad y representación del tiempo, implica que **tardis.js** no conoce la unidad de tiempo o como parsear dicha unidad a un formato humano o como interpretarlo en el formato que este en el set de datos, en su lugar, se delega dicha funcionalidad a unas funciones que se toman como parámetros de configuración, tanto el parseo de las fechas/tiempos en el set de datos, como su representación a un formato legible por el usuario. Así que, para casos sencillos, como un set de datos donde el tiempo este en años, las funciones de parseo y formateo son sumamente sencillas; y para casos donde se trabaje con fechas más complejas (por ejemplo un set de datos con los eventos del JFK), se puede usar una librería JavaScript aparte para lidiar con dicha tarea. Si no se concreta dichas funciones en los parámetros a la hora de invocar al *widget* inicialmente, entonces se parsea como un numero simple en años, y se representa con una notación simple en que el caso de ser una fecha negativa, se agrega " b.C" al final para remarcar que es una fecha anterior al año cero.

```
    parser   : function (value) { return value; }, // Parsing function for input
time values
    formatter: function (value /*, where, original*/ ) { // Formating function
for user presentation of time
      if (value < 0) {
        return String(-value) + ' b.C.';
      } else {
        return String(value);
      }
    },
```

Código 4: Funciones de parseo y formateo usadas por defecto

La función de formateo, toma dos parámetros opciones con los que se obtiene desde donde se esta llamando (útil para el formateo de las marcas en el *timeline*), y el valor original en el set de datos.

4.1.3 jQuery

En el desarrollo del prototipo, se opto por usar la casi omnipresente librería jQuery y luego se observo que seria adecuado convertir el *widget* en un plug-in de jQuery, con lo que se simplifica su uso y se integra mejor con la librería. Para conseguir que sea un plug-in de jQuery, primero hay que encerrar el código en una función que es ejecutada en el lugar y que toma el objeto **`jQuery`** como parámetro, permitiendo hacer que el código del plug-in pueda ser menos conflictiva, ya que se puede dar el caso que al usar otras librerías JavaScript la variable **`$`** no sea siempre el objeto **`jQuery`**. Esta parte del código es realizada por los ficheros **`tbegin.js`** y **`tend.js`**.

Luego se procede asignar a **`$.fn.TardisTL`** una función que toma dos parámetros: **`data`** y **`options`**, que se encargara de crear el objeto del *widget* y/o llamar a los métodos del objeto creado. Con esto conseguimos añadir un método a cualquier selector jQuery para que invoque nuestra función. También asignamos el constructor y exponemos los valores por defecto de configuración.

```
+function ($) { "use strict";
  // Codigo ...
  // ...

    $.fn.TardisTL = function (data, option) {
      //return new TardisTL($(this), data, option);
      var ret   = -1;
      var jq    = this.each(function () {
        // Function
        var $this   = $(this);
        var tn      = $this.data('tardistl');
        var action  = data;

        if ($.isArray(data) && !tn) { // Missing data
          // Stores the Tardis object in the element
          $this.data('tardistl', (tn = new TardisTL($this, data, option)));

        } else if (typeof action === 'string') { // Public methods
          switch (action) {
            case 'setEntry' :
              tn.setEntry(option);
              break;

            case 'setView' :
              tn.setView(option);
              break;

            case 'setZoom' :
              tn.setZoom(option);
              break;

            case 'redraw' :
              tn.update();
              tn.relocateFlags();
              tn.update();
              break;

            case 'index' :
              ret = tn.index;
              break;

            default:
              break;
          }

        } else if (typeof action === 'number') { // Go to data index X
          tn.setEntry(data);
        }
```

```
      });

      if (ret === -1) { // Return jQuery object or method return value
        return jq;
      }
      return ret;

    };

    $.fn.TardisTL.Constructor = TardisTL;
    $.fn.TardisTL.DEFAULTS = TardisTL.DEFAULTS; // Exposes TardisTL defaults to
the world. WARING, NOT EDIT IT DIRECTLY


}(jQuery);
```

Código 5: Creación del plug-in

Como se puede ver en código 5, se usa el método data de jQuery para almacenar el objeto del *widget* en el nodo DOM donde vamos a construir el *widget*. Si no se encuentra el objeto, entonces se crea y se le pasa como parámetros, `data` y `options`, aparte del nodo DOM donde se va a construir el *widget*. Así que para crear un *widget* habría que invocarlo como "`$(selector).TardisTL(datos, opciones);`". Si el objeto ya existe y el parámetro pasado, es una cadena de texto, entonces se interpretara que se esta llamando a algún método del *widget* y por lo tanto se ejecutara la acción necesaria. Finalmente si el objeto ya existe y el parámetro es un número, entonces se interpreta que se quiere seleccionar la entrada cuyo índice es dicho número.

4.1.4 Algoritmo de colocación

El problema de colocar las banderas para evitar los solapamientos no es tan trivial como podría parecer. A priori podemos pensar que se trata de un problema de la colocación óptima de rectángulos en un espacio finito (problema de "bin packing 2D"). Sin embargo este caso tiene ciertas restricciones que afortunadamente nos permite simplificar el problema, pues las banderas se tienen que colocar del tal forma que su lado izquierdo tenga una posición horizontal precisa, que coincide con el índice de tiempo del evento que representa.

El algoritmo usado parte de que set de datos esta ordenado por fecha, lo cual implica que estén ordenados por su futura posición horizontal, y se recorre en dicho orden. A medida que se recorre el set de datos, se va colocando cada bandera de cada entrada en la primera fila donde se calcule que no ocurre solapamiento. Cada fila almacena el valor de la posición donde termina la caja de la ultima bandera insertada, así que cada vez que se inserta una bandera en una fila, se obtiene el valor

de `right` del `<div>` de la bandera y lo almacena en un array según la fila en que se inserte. Dicho array contendrá la "anchura" de cada fila. Para ver en que fila se inserta, se recorren por orden el array de anchuras de las filas, y se inserta en la primera fila en que la anchura sea menor que la posición que tendría la bandera.

Además se permite crear nuevas filas si una bandera se va a solapar con las pre-existentes. Si no se puede crear una nueva fila (por que se llegue al limite impuesto), entonces se pone en la fila en la que se solape lo menos posible.

En resumen, se trataría de un algoritmo de tipo *first-fit*.

Para la colocación horizontal, se calcula un valor llamado PPTU (Pixels Per Time Unit), indica cuantos pixels mide una unidad de tiempo. Se calcula dividiendo el ancho del `<div>` contenedor, por el rango de tiempo de la ventana de tiempo visible. Cuando se hace "zoom" este valor se re calcula, aplicando las nuevas posiciones y tamaños a todas las banderas.

4.1.5 Hammer.js - Eventos táctiles

Hammer.js [8] se trata de una librería compatible con jQuery que permite tratar eventos táctiles en el navegador de la misma forma que se tratan los eventos normales como un *click* o arrastrar. Ante el requisito del cliente de que el *widget* debe de funcionar en dispositivos táctiles, particularmente tabletas y móviles, se hace imperativo buscar algún método de simplificar el código de dicha clase de eventos y por ello se opto por usar hammer.js, particularmente su versión como plugin de jQuery.

Al usar la versión que es un plugin de jQuery, podemos detectar fácilmente si se esta presente, por lo tanto convirtiendo esta librería en algo opcional de cara al cliente. Al ser un plugin, los selectores de jQuery ganan un método "`hammer(opciones)`". Por lo tanto, haciendo un simple "`if (typeof this.$root.hammer !== 'undefined')`" detectamos la presencia de hammer.js .

Hammer.js permite, usando el método ".on" de jQuery, controlar estos eventos :

- hold
- tap
- doubletap
- drag, dragstart, dragend, dragup, dragdown, dragleft, dragright
- swipe, swipeup, swipedown, swipeleft, swiperight

- transform, transformstart, transformend
- rotate
- pinch, pinchin, pinchout
- touch
- release
- gesture

A nosotros, particularmente, nos interesa los eventos de drag, dragstart y dragend para deslizar la ventana; pinch, pinchin y pinchout para realizar "zoom" sobre la ventana.

Aparte de definir estos eventos, hammer.js modifica los datos que lleva el objeto "`event`" que recibe el *handler* del evento, añadiendo una entrada "`gesture`" con varios campos propios (véase https://github.com/EightMedia/hammer.js/wiki/Getting-Started#event-data). A nosotros nos interesa particularmente los subcampos "`center.pageX`" y "`center.pageY`", que nos dan las coordenadas del centro del evento táctil. También se añade un campo "`type`" que indica que tipo de evento táctil ha lanzado el evento, lo cual nos permite usar un mismo *handler* para varios eventos distintos y poder ejecutar el código adecuado en cada caso.

```
      // Tactile events for main surface
      this.$root.hammer().on('dragstart dragright drag dragend pinchin
pinchout',
                             '.tardistl-container', $.proxy(function (event) {
        event.stopPropagation();
        event.gesture.preventDefault();
        switch (event.type) {
          case 'dragstart':
            this.dragging = true;
            this.initialPan = [event.gesture.center.pageX,
event.gesture.center.pageY];
            break;

          case 'drag':
            var deltaX = (event.gesture.center.pageX - this.initialPan[0]) /
this.PPTU ;
            var deltaY = (event.gesture.center.pageY - this.initialPan[1]) ;
            this.dragRepaint(deltaX, deltaY);
            this.initialPan = [event.gesture.center.pageX,
event.gesture.center.pageY];
            break;

          case 'dragend':
            this.dragging = false;
            break;

          case 'pinchin':
            if (! this.dragging) {
              this.setZoom(this.zoom * 2);
              event.preventDefault();
            }
            break;

          case 'pinchout':
            if (! this.dragging) {
              this.setZoom(this.zoom * 0.5);
              event.preventDefault();
            }
            break;

          default:
            break;
        }

        return true;
      }, this));
```

Código 6: Código que usa Hammer.js para implementar el deslizamiento y el zoom con eventos tactiles

Como se puede observar en Código 6, al inicio de un deslizamiento, guardamos las coordenadas del evento y activamos una bandera. Luego, según va viniendo cada evento “drag”, se calcula el desplazamiento y se solicita repintar las banderas en su posición correspondiente con el método “`dragRepaint`”. Finalmente cuando termina el evento, se desactiva la bandera.

4.1.6 Problemas de rendimiento

Durante el desarrollo se ha trabajado principalmente sobre el navegador Chrome, realizando aparte pruebas puntuales sobre Firefox, Safari y Internet Explorer 8, 9 y 11. Con el navegador Chrome no se detecto problemas de rendimiento o suavidad con los ejemplos de prueba, sin embargo si que se pudo apreciar ciertos problemas de suavidad usando Firefox y Internet Explorer cuando la cantidad de elementos empieza a ser alta (≥ 90). Adicionalmente se aprecio diferencias de rendimiento de un mismo navegador en la misma maquina, dependiendo del sistema operativo. Por ejemplo en un Tablet PC, que usa una gráfica integrada AMD, Firefox rinde notoriamente peor en GNU/Linux que en Windows 8.

Para tratar de localizar el origen del problema de rendimiento y suavidad, se realizaron varios test de *profiling* sobre el *widget* con las herramientas que incluyen Firefox y Chrome. Con ellas se pudo observar que aunque Firefox es efectivamente más lento re-dibujando o re-colocando los elementos, la diferencia medida no es tal, como para justificar la sensación de falta de suavidad percibida.

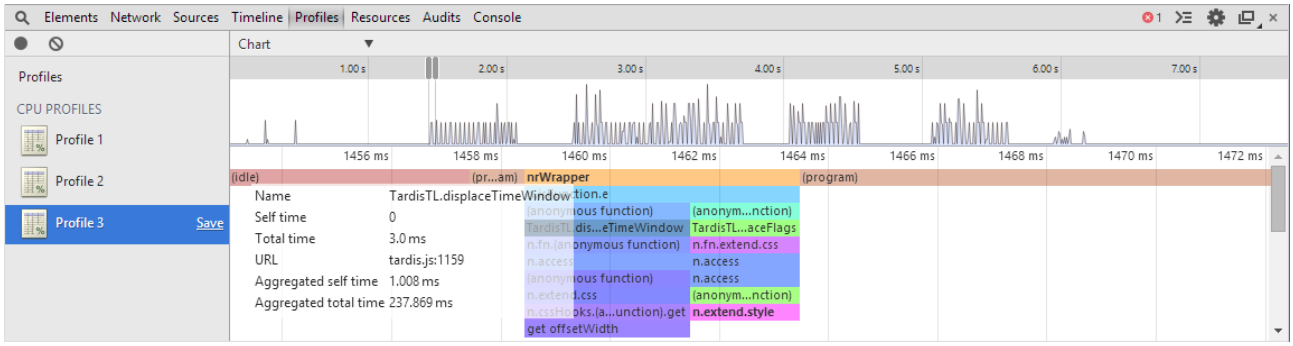

Figura 11: Profiling con Chrome

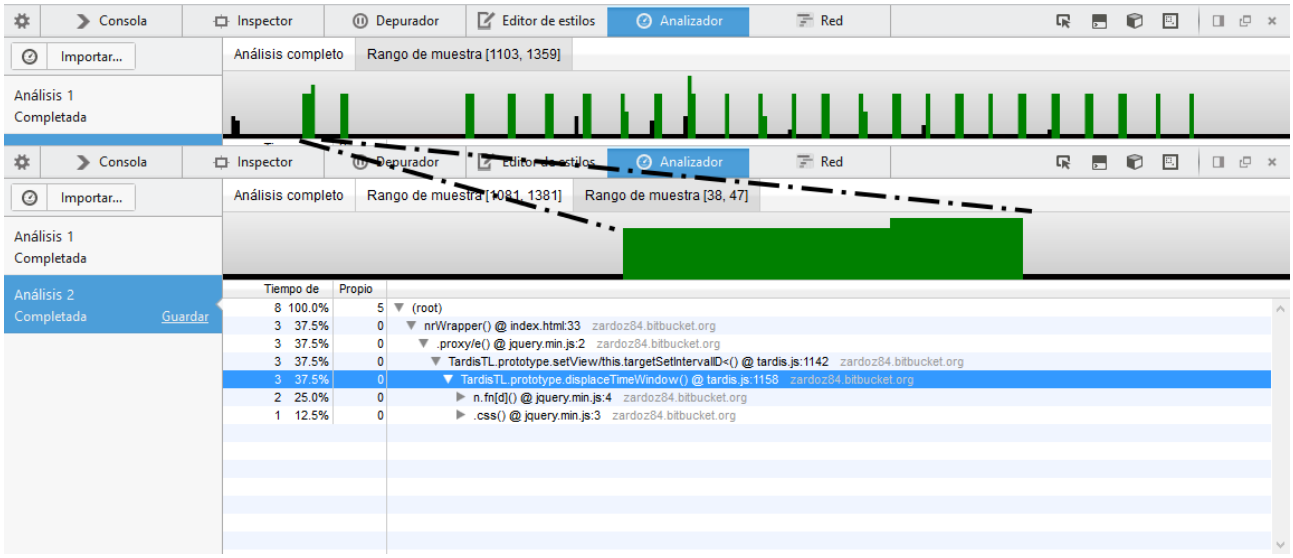

Figura 12: Profiling con Firefox

Pruebas adicionales que se hicieron con algunos test de eventos de JavaScript mostraron que esta situación no es única del *widget* y se pueden apreciar en otras *widgets* o simplemente dibujando en un `<canvas>` con JavaScript desde eventos de ratón.

Finalmente se opto por probar a hacer el re-dibujado/re-colocación de los elementos usando la función especial "`window.requestAnimationFrame`". Dicho función toma como parámetro una función que es ejecutada una sola vez, cuando el navegador determina que es adecuado para realizar una animación suave. Esto es análogo a esperar el retrazado vertical en el mundo de los videojuegos. En Chrome, el efecto es inapreciable, lo cual induce a pensar que dispara sus eventos de ratón de forma sincronizada a cuando lo haría esta función. En Firefox y IE 11, el efecto es apreciable. La sensación de suavidad mejora notoriamente y se hace aceptable. Sin embargo este método no sirve para el caso de Internet Explorer 8 y 9, aunque se realizaron ciertos cambios en el código para tratar de optimizar el re-colocado/re-dibujado de los elementos que consiguió que dichas versiones de Internet Explorer funcionen con suavidad. Todos estos cambios de mejora de rendimiento y el uso "`window.requestAnimationFrame`", se hace mediante el método "`dragRepaint`".

```
    /**
     * Does the apropiate repaint when the user drags the view
     * @param deltaX Displazament in Abstract termporal Units
     * @param deltaY Vertical displazament in pixels (scroll)
     */
    TardisTL.prototype.dragRepaint = function (deltaX, deltaY) {
      if (this.dragging && ! isNaN(deltaX) && deltaX !== 0) { // Drag on
horizontal
        var animate = isIE || isFirefox;
        this.setView(this.view_cursor - deltaX, animate); // Invert direction
      }

      if (this.using_scrollbar && ! isNaN(deltaY) && deltaY !== 0 ) { // Drag on
vertical
        var value = this.scrollbar.slider("value") + deltaY;
        value = Math.min(this.scrollbar.slider("option", "max"), value);
        value = Math.max(this.scrollbar.slider("option", "min"), value);
        this.scrollbar.slider("value", value);
        this.offset = value;
        if (typeof (requestAnimationFrame) !== 'undefined') {
          if (this.targetScrollSetIntervalID === 0) {
          this.targetScrollSetIntervalID = requestAnimationFrame($.proxy(
            function () {
              this.emplaceFlags(false);
              this.targetScrollSetIntervalID = 0;
            }, this));
          }

        } else {
          this.emplaceFlags(false);
        }
      }

      return false;
    };
```

Código 7: Método dragRepaint

Otros trucos que se probaron sin apreciarse apenas diferencias, fue forzar a que el `<div>` que contiene las banderas sea acelerado por la GPU, usando la propiedad CSS "`transform: translateX`" en vez de usar la propiedad "`left`" para el posicionamiento.

```
    if (oldIE) {
      this.events_container.css({
        left: String(left) + 'px',
      });
      this.int_cont.css({
        left: String(left) + 'px',
      });
    } else { // Sane web browser code goes here

      this.events_container.css({
        'transform': 'translateX(' + left.toString() + 'px )',
        '-ms-transform': 'translateX(' + left.toString() + 'px )',
        '-webkit-transform': 'translateX(' + left.toString() + 'px )',
      });
      this.int_cont.css({
        'transform': 'translateX(' + left.toString() + 'px )',
        '-ms-transform': 'translateX(' + left.toString() + 'px )',
        '-webkit-transform': 'translateX(' + left.toString() + 'px )',
      });
    }
```

Código 8: Ejemplo de uso de transform:translate

4.1.7 Parámetros de configuración

Ante las diversas opciones planteadas por el cliente y con el deseo expreso de que sea flexible para poder usarlo en una variedad de situaciones, se ha implementado un flexible sistema de opciones y parámetros mediante un objeto JavaScript. Para ello se usa el método "`$.extend`" de jQuery que permite "extender" un objeto con nuevas entradas o remplazarlas. Dicho de otra forma, si el objeto que se recibe como parámetros de configuración, se omiten una o varias entradas, entonces "`$.extend`" permite usar los valores de un objeto que contiene los valores por defecto.

```
  /**
   * Base class constructor
   * @param id jQuery object were to place the widget
   * @param data array of data to be used
   * @param option optional options
   */
  var TardisTL = function (id, data, option) {

    ...

    // Default values
    this.options = $.extend({}, TardisTL.DEFAULTS, typeof option === 'object' &&
option);
```

Código 9: Llamada a $.extend

Las opciones presentadas en el *widget* se detallan en el anexo, pero se detallan aquí las más importantes :

- **min_zoom** y **max_zoom** : Niveles máximos y mínimos de zoom. Si se omite, se tratan de auto determinar a partir del set de datos.
- **parser** : Función de parseado de las fechas del set de datos a un número. Dicha función debería de ser biyectiva para el dominio del set de datos.
- **formatter** : Función de formateado de las fechas, a partir de la representación numérica.
- **minimap** : Activa/desactiva un minimapa de tiempo. El minimapa muestra todo el intervalo de tiempo y un selector deslizable con el tamaño a escala de la ventana de tiempo visible.
- **marks** : Objeto que define cuando y a que nivel de zoom aparecen diversas marcas de tiempo en el *timeline*. Esto define las marcas de milenio, centuria, 50 años, décadas, etc... Y ademas permite controlar si dichas marcas abarcan todo el *timeline* o solo en determinadas fracciones del *timeline*, una cualidad útil si el set de datos presenta una zona de gran densidad de entradas.
- **minimap_marks** : Análogo a “marks”, pero en el minimapa, aunque se omite (obviamente) el control de visibilidad por nivel de zoom.

4.2 tardis.js

Como se menciono previamente, se decidió implementar por separado la funcionalidad que une el *timeline*, con un panel tipo carrusel, botones de navegación y un mapa. En los ficheros fuente, esta

bajo `/src/js/tardis-wraper.js` y el resultado de unir dicha funcionalidad con tardis-tl.js, es la librería **tardis.js** .

Aparte de implementar un carrusel propio y usar el *widget* Leaflet map para el mapa, contiene código que los mantiene sincronizados y ofrece unas cuantas opciones de layout de los tres elementos, uno de ellas usando jQuery UI Layout para crear un layout modificable por el usuario final.

4.2.1 Carrusel

En el desarrollo del prototipo, se uso en un principio Bootstrap para crear el layout y también se uso su implementación de carrusel con buen éxito. Sin embargo, ante el deseo expreso del cliente de que no debería de ser obligatorio usar Bootstrap, entonces se vio la necesidad de implementar un carrusel propio, que fuese liviano.

Para implementar nuestro propio carrusel, se observo como funciona el de Bootstrap, donde en esencia tenemos varios `<div>` ocultos en la misma ubicación, y solo uno de ellos es visible (el elemento seleccionado). Para cambiar de panel, solo hay que ocultar el actualmente visible, y hacer visible el que se desea. A esta funcionamiento básico, se le añade unos simples botones de navegación para pasar al anterior o al siguiente elemento. A diferencia de la implementación de Bootstrap, nuestra implementación no usa animaciones, lo cual conlleva que el usuario tenga una mayor sensación de rapidez al usarlo.

```
<div class="panels">
  <a class="panel-control left" data-slide="prev" href="#carousel"></a>
  <a class="panel-control right" data-slide="next" href="#carousel"></a>
  <article class="item active"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
  <article class="item unactive"></article>
```

Figura 13: Ejemplo del simple árbol DOM del carrusel

Un problema planteado por esta solución es que si hay muchos elementos con un cuerpo grande, podría haber problemas de uso de memoria RAM en dispositivos como móviles o tabletas. Para tratar de aliviar la situación, al menos inicialmente, se añadió una opción, en la que si se detecta que el campo del set de datos que seria el contenido del panel es una URL, entonces se trata de realizar una carga asíncrona de dicha URL. Lo que se obtenga, se mostrara directamente en el contenido del panel. Así, inicialmente no se tiene todo los contenidos en memoria y cuando el usuario final cambie de panel visible, entonces se procede a cargar su contenido de la URL remota. De esta forma, intercambiamos velocidad inicial por uso de RAM.

Este método es mejorable, pues los paneles que se han cargado no se descargan y por lo tanto el uso de RAM va aumentando según el usuario vaya cargando más paneles. Una posible mejora seria que descargase los paneles que se vieron hace mayor tiempo.

Para detectar si es una URL, en vez de pasar una compleja expresión regular [9] que detecte con seguridad si es una URL, y que por lo tanto sera lenta para contenidos largos, se opto por simplemente verificar si se contiene algún carácter no valido para URLs en el contenido. Es decir, si se detecta un espacio o alguno otro carácter extraño, entonces no estamos ante una URL y se carga de forma normal el panel. Si no, entonces se interpreta que es una URL y se intentara usara la carga asíncrona. Este método esta lejos de ser perfecto, ya que podria dar falsos positivos si el contenido del panel es solo una palabra, pero en tal caso, ¿Realmente se necesita la carga asíncrona si los contenidos a presentar son tan breves como una simple palabra ?

4.2.2 Leaflet map y Leaflet cluster

Uno de los principales requisitos del cliente, y por lo que se observo inicialmente TimeMapper, es que aparte de poder presentar una lista de Polígrafos por orden cronológico, era poder ver en un mapa su lugar de nacimiento o residencia. TimeMapper usa Leaflet como motor de mapas. Se hizo pruebas con dicho motor y resulto adecuado para lo que el cliente desea. Por lo tanto se procedió a usarlo en el prototipo y en la versión final.

Figura 14: Widget para mapas Leaflet

A pesar de que el *widget* de mapas funciona bien, se observo un pequeño problema con algunos set de datos. Si muchas entradas coinciden en el mismo lugar, resulta bastante complicado poder seleccionar la entrada deseada en el mapa. Como el cliente deseaba una solución a dicho problema, se encontró un plug-in para Leaflet, llamado Leaflet cluster. Como indica su nombre, crea *clusters* de marcas en el mapa, y según el nivel de zoom en el mapa, los despliega en una espiral. Este plug-in no es una dependencia real de la librería, aunque se ha usado en todos los ejemplos mostrados y es bastante recomendable.

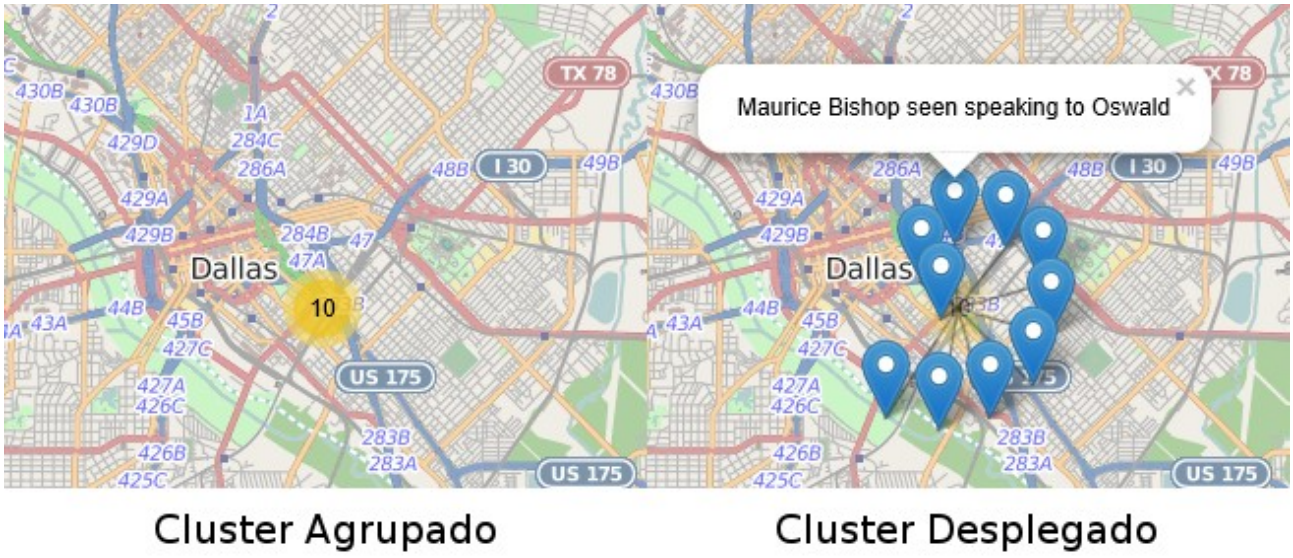

Figura 15: Clusters del plugin Leaflet Cluster

4.2.3 Opciones de layout y Jquery UI Layout

En el prototipo inicial se usó Bootstrap para crear el layout inicial y poder crear el código que sincroniza los *widgets*. Sin embargo, a la hora de integrarlo con el producto del cliente, se vio que producía ciertos efectos indeseados, así que primero se elimino la dependencia de Bootstrap, y se implemento un método para poder hacer agnóstico al sistema de layout. Además, el cliente expreso que deseaba la funcionalidad de que el usuario final pudiese ajustar el tamaño de los *widgets* para conseguir mayor comodidad. Este requisito fue resuelto usando jQuery UI Layout, el cual es un sistema de paneles que permiten ser dimensionados por el usuario o incluso ocultar los paneles, usando unos tiradores.

Para poder elegir si se va a usar jQuery UI Layout o se quiere usar algún sistema alternativo de layout, se añadió una opción "`ui_layout`" que permite elegir tres modos.

- "`jq_ui_layout`" : Es la opción por defecto y permite usar jQuery UI Layout
- "`external`" : Junto con tres parámetros de las opciones, permite usar selectores de jQuery para concretar donde se va a construir los *widgets* en la página. El ejemplo de "23F" usa este sistema para crear el layout con Bootstrap.
- "`none`" : No usa ningún sistema de layout especial, simplemente se limita a crear un para cada *widget* y se deja al desarrollador de la página, que aplique el layout que dese mediante CSS y/o JavaScript.

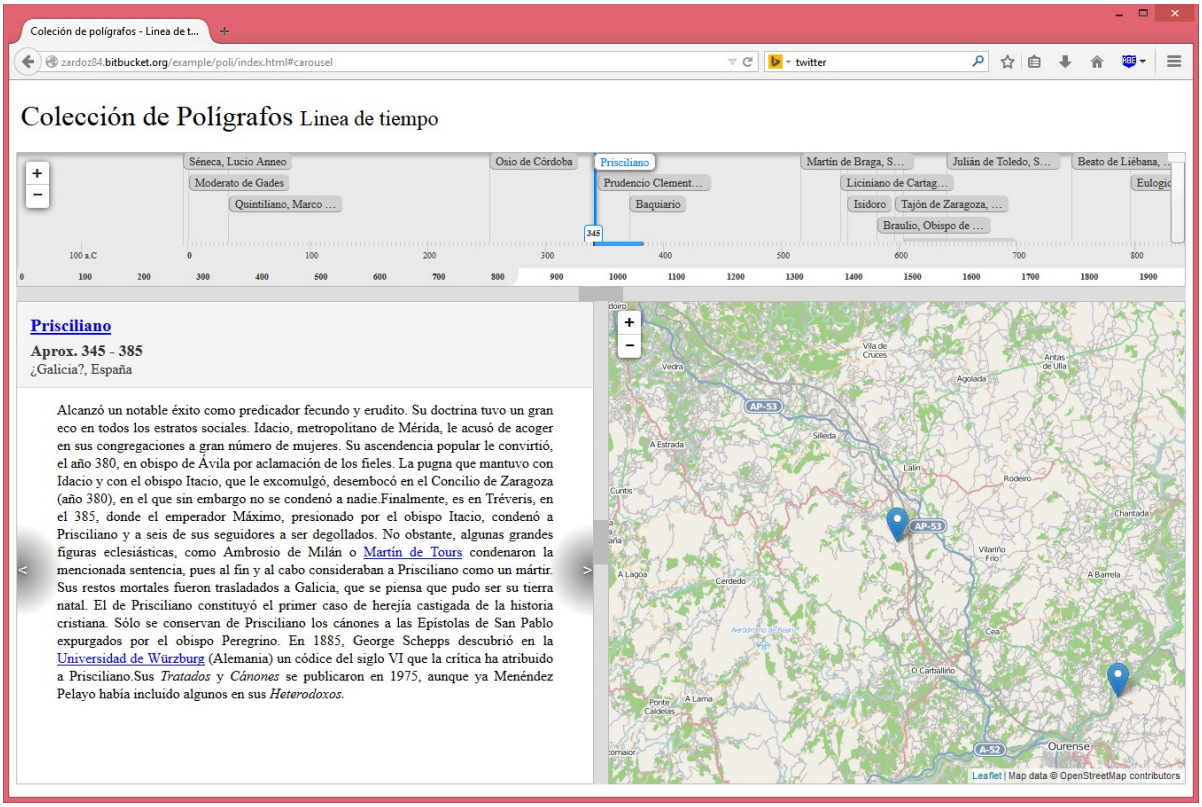

Figura 16: Ejemplo del layout por defecto con jQuery UI Layout

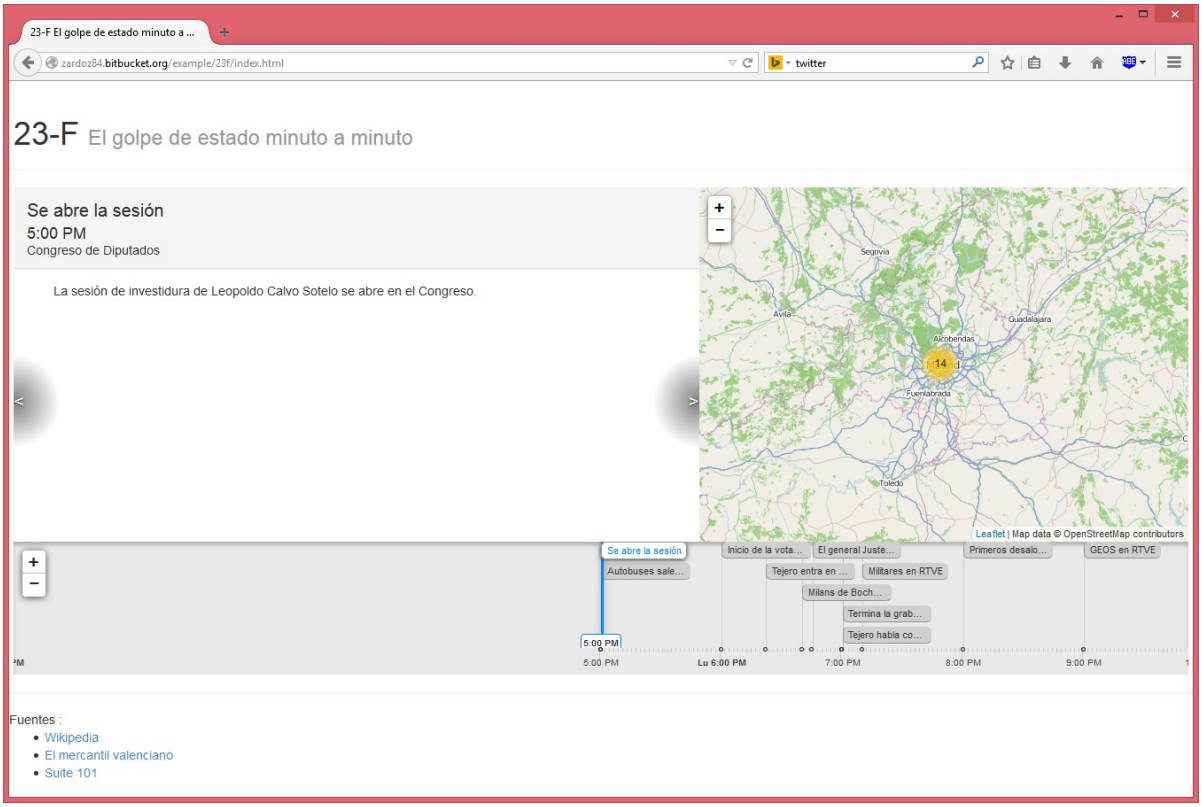

Figura 17: Ejemplo de Layout alternativo con Bootstrap

Si se está usando jQuery UI Layout y además se está usando la opción de “fixed_height”, es

necesario usar el evento de Tardis TimeLine “resize” para cambiar el tamaño del panel que lo contiene, para que pueda mostrar correctamente el *widget*.

4.2.4 Sincronización de los widgets

Para la sincronización de los *widgets*, se ha usado principalmente los eventos que dispone el *widget* Tardis TimeLine y Leaflet. El carrusel al ser una solución interna propia, en el mismo código que cambia de ficha en los botones, contiene el código para seleccionar la entrada adecuada en el *timeline* y centrar el mapa en la posición geográfica de la entrada.

Entonces lo primero que se realiza después de crear los tres *widgets*, es recorrer el set de datos para ir creando los paneles del carrusel y las marcas en el mapa de Leaflet. Cada marca del mapa, lleva anotado el índice de la entrada del set de datos que le corresponde y una auto referencia al objeto que es el *widget*, lo cual permite llamar al código interno para precargar el contenido de la entrada, si es necesario, seleccionar el panel que tiene que ser visible y llamar al *widget* de Tardis TimeLine para indicarle cual bandera tiene que estar seleccionada.

```
    // Fill carousel panels and map marks
    for (var n=0; n < data.length; n++) {
      var entry = data[n];
      // Build Entry title with dates and place
      var str = '<article class="item unactive">' +
                '<div class="panel-heading"><h2>' + entry.title;

      // Apply format to Start/End and writes it to the header
      entry['start_str'] = this.options.formatter(entry.start);
      str = str + '<br/></h2><h3>';

      if (entry['end']) {
        entry['end_str'] =  this.options.formatter(entry.end);
        str = str + this.options.formatter_int(entry.start, entry.end);
      } else {
        entry['end_str'] = "";
        str = str + entry.start_str;
      }
      str = str + '</h3>\n';

      // Put place if there is any
      if (typeof entry.place !== 'undefined' ) {
        str = str + entry.place + '\n';
      }
      str = str + '</div>\n';

      // Build Main body of text
      str = str + '<div class="panel-body">';
      if (! this.options.dynamic_load) {
        str = str + entry.description;
        entry.description = null; // We not need this now, so we free ram
      }
      str = str + '</div></article>\n';

      this.panels.append(str); // Appends each page

      // Sane location or fallback location
      var loc = this.options.fallback_location;
      if (typeof entry.location === 'object' && entry.location.length >= 2 &&
typeof entry.location[0] === 'number' && typeof entry.location[1] ===
'number'  ) {
        loc = entry.location;
      } else {
        entry.location = this.options.fallback_location; // Enforce to mark it
as a entry without location (array with length 0)
      }

      // Add markers if there is a valid location (not empty array)
      if (loc.length >= 2 ) {
```

```
        var marker = L.marker(loc).addTo(markers);

        if (typeof entry.map_marker_title === 'undefined') { // Try to get a
title for the popup
          marker.bindPopup(entry.title);
        } else {
          marker.bindPopup(entry.map_marker_title);
        }

        marker['slide'] = n; // Self reference to entry slide
        //marker['entry'] = entry;
        marker['self'] = this; // Hackish way to carry a self reference

        // Dbl. Click event in a map marker
        marker.on('click', $.proxy(function() {
          // Sync map, tardistl and carousel
          this.self.loadEntryData(this.slide);

this.self.panels.children('.item').removeClass('active').addClass('unactive');

this.self.panels.children('.item').eq(this.slide).removeClass('unactive').addCl
ass('active');
          this.self.tardistl.TardisTL(this.slide);
        }, marker));
      }

    }
```

Código 10: Generación de paneles y marcas del mapa. Se destaca el código que actualiza el carrusel y Tardis TimeLine desde el mapa.

En el código que se ejecuta cuando se hace *click* en los botones de siguiente y atrás, el código cambia el panel que es visible y procede a llamar al *widget* de Tardis TimeLine para que cambie la bandera que esta seleccionada. Luego, procede a centrar la vista en la marca del mapa que corresponde a la entrada actualmente seleccionada.

```
    // Add event to sync map and tardistl when we change slide with nav buttons
    this.panels.find(".panel-control[data-slide='prev']").on('click',
$.proxy(function () {
      var index = this.panels.children('.active').index('.panels .item') -1 ;
      if (index < 0) {
        index = this.data.length -1;
      }

      var entry = this.data[index];
      var zoom = this.map_obj.getZoom();
      // Updates map and tardistl
      this.loadEntryData(index);
      if (entry.location.length >= 2) {
        this.map_obj.setView(entry.location, zoom);
        this.no_location.hide();
      } else {// Display message of not know location
        this.no_location.show();
      }
      this.tardistl.TardisTL(index);

      this.panels.children('.item').removeClass('active').addClass('unactive');

this.panels.children('.item').eq(index).removeClass('unactive').addClass('activ
e');

    }, this));

    this.panels.find(".panel-control[data-slide='next']").on('click',
$.proxy(function () {
      var index = this.panels.children('.active').index('.panels .item') +1 ;
      index = index % this.data.length;

      var entry = this.data[index];
      var zoom = this.map_obj.getZoom();
      // Updates map and tardistl
      this.loadEntryData(index);
      if (entry.location.length >= 2) {
        this.map_obj.setView(entry.location, zoom);
        this.no_location.hide();
      } else {// Display message of not know location
        this.no_location.show();
      }
      this.tardistl.TardisTL(index);

      this.panels.children('.item').removeClass('active').addClass('unactive');

this.panels.children('.item').eq(index).removeClass('unactive').addClass('activ
e');
```

```
    }, this));
```

Código 11: Botones de siguiente y atrás

En el evento "`tardistl.flagn`" que se dispara después de que se haya cambiado la entrada seleccionada en el mismo, análogamente se llama al código interno para precargar el contenido de la entrada, se cambia el panel que tiene que ser visible y finalmente se centra la vista del mapa en la marca que le corresponde a la entrada.

```
    // Add event to sync map and carousel when we change selected element in
tardistl
    this.tardistl.on('tardistl.flagn', $.proxy(function(evt, index) {
      this.loadEntryData(index.newindex);
      this.panels.children('.item').removeClass('active').addClass('unactive');
this.panels.children('.item').eq(index.newindex).removeClass('unactive').addCla
ss('active');
      var zoom = this.map_obj.getZoom();

      if (this.data[index.newindex].location.length >= 2) {
        this.map_obj.setView(this.data[index.newindex].location, zoom);
        this.no_location.hide();
      } else {// Display message of not know location
        this.no_location.show();
      }

    }, this));
```

Código 12: Evento tardistl.flagn

4.2.5 Parámetros de configuración

Se aplica el mismo sistema que en el *widget* de Tardis TimeLine. El listado de las opciones se presenta en el anexo, pero se procede a mostrar aquí, un listado de las mas interesantes :

- **leaflet** : Es un objeto que pasa al *widget* Leaflet varias opciones de configuración.
    - **tilemap_url** : URL de la fuente de teselas usada por el mapa. Por defecto se usa la Open Street Map
- **ui_layout** : Control del sistema de layout antes mencionado.
- **formatter** : Función de formateado de las fechas, a partir de la representación numérica. Puede ser distinta a la usada en Tardis TimeLine
- **formatter_int** : Función de formateado a usar cuando se quiere mostrar un intervalo de

tiempo.

- **tardistl_container** , **map_container** y **carousel_container** : Selectores de jQuery donde construir los *widgets* si ui_layout es “external”.
- **dynamic_load** : Intenta la carga dinámica de los contenidos de cada entrada si detecta que son una URL.
- **fallback_location** : Coordenadas a usar si una entrada del set de datos no posee una ubicación geográfica asociada. Si se dejase como un array vació, entonces en vez de asociar una ubicación geográfica, se mostrara un aviso de que no hay ubicación conocida.

5. Pruebas

Durante el desarrollo se ha procedido a probar los *widgets*, usando varios set de datos en múltiples navegadores y dispositivos distintos. Particularmente se ha probado en :

- iPad 1 : Safari y Chrome para iPad
- Huawey G510 (Android 4.1) : Navegador por defecto y Chrome para Android
- PC Core 2 Duo con gráfica Intel y 2 GiB de RAM.
    - Kubuntu 14.04 LTS 32 bits
        - Firefox
        - Chrome
        - Rekonq
    - Windows XP SP3
        - Internet Explorer 8
        - Firefox
        - Chrome
        - Safari
- PC AMD FX-4100 quad core, con gráfica NVIDIA GTX 680 y 8 GiB de RAM
    - Kubuntu 14.04 LTS 64 bits , driver propietario

        - Firefox
        - Chrome
        - Rekonq
    - Windows 8.1 Profesional 64 bits
        - Firefox
        - Chrome
        - Internet Explorer 11
- HP Tablet PC Touch Smart TX2-1350 , con 4 GiB de RAM y gráfica Radeon HD 3200
    - Kubuntu 14.04 LTS 64 bits , driver gráfico libre
        - Firefox
        - Chrome
        - Rekonq
    - Windows 8.1 Profesional 64 bits
        - Firefox
        - Chrome
        - Internet Explorer 11

Se ha usado tres ejemplos con set de datos distintos y que también tienen distintas opciones.

- **Polígrafos** : Muestra un listado de Polígrafos obtenidos de la colección de la Fundación Ignacio Larramendi. Consiste en aproximadamente 90 entradas, y usa en su mayoría, las opciones por defecto de Tardis.js. Algunas entradas han sido alteradas para provocar que se ejecuten casos especiales de entradas sin ubicación o carga dinámica, etc.
- **23F** : Resume varios eventos que acontecen durante el fallido golpe de estado. Usa un set de datos mas reducido y donde los eventos transcurren dentro de una franja de apenas 24 horas. Actúa también como ejemplo de uso de otro layout distinto a jQuery UI Layout, usando Bootstrap.
- **JFK** : Basado en los datos que muestra la demo de SIMILE Timeline de JFK, muestra un set

de datos grande en que la densidad de los eventos no es uniforme. Además, en el set de datos, se barajan eventos que están en algunos casos separados en meses y otros en unos pocos minutos, actuando como ejemplo perfecto de como establecer marcas no uniformes a lo largo de toda la franja de tiempo.

5.1 iPad 1

Las pruebas realizadas en el iPad 1 han sido satisfactorias si tenemos en cuenta que este dispositivo es bastante limitado para los estándares actuales. La presentación es correcta en ambos navegadores y la suavidad percibida por el usuario es aceptable para su uso.

5.2 Móvil Android Huawei G510

Este móvil de gama media baja de finales del 2013, es capaz de hacer funcionar correctamente el *widget*, sin embargo tiene problemas para poder mover con suavidad aceptable un *timeline* con una cantidad mediana o grande de entradas (demos de Polígrafos y JFK). Además, en este tipo de pantalla, el uso de jQuery UI Layout no es recomendable ya que difícilmente se puede aprovechar bien la pantalla. El uso de una layout *reponsive* con Bootrstrap da unos resultados mucho mejores (demo de 23F).

5.3 PC Core 2 Duo

Las pruebas realizadas sobre la *workstation* que se ha usado durante la mayor parte del tiempo, han sido de importancia, ya que se usa como equipo referencia de un PC de características normales que pueda usar los usuarios finales del cliente.

En este caso, el navegador Chrome, y por extensión los navegadores que usan WebKit como Safari y Rekonq, han tenido una respuesta excelente en ambos sistemas operativos, con todos los set de datos. Sin embargo Firefox presenta un aparente rendimiento subpar respecto a Chrome, sobretodo en GNU/Linux. Nosotros le atribuimos dicho problema a que Firefox podría no aprovechar bien del todo, la aceleración de la tarjeta gráfica, en especial en GNU/Linux, y quizás a la frecuencia con la que se disparan los eventos de “mousemove” y “drag”

Las pruebas con Internet Explorer 8, han sido una sorpresa, ya que nunca se ha marcado como objetivo conseguir que el comportamiento fuera mas allá de funcionar correctamente en Internet Explorer 8, hasta el punto en que su presentación es obviamente mas espartana. Nos esperábamos un peor comportamiento de este navegador, sin embargo, presenta una suavidad bastante decente e incluso mejor que Firefox, teniendo en cuenta la obsolescencia de este navegador. Sospechamos que

esta relacionado a que no se aplican ciertos efectos CSS como los bordes redondeados, transparencias y sombras.

5.4 PC AMD FX-4100

Este PC representa a lo que seria un PC de gama media actual. En este caso, el rendimiento es más que satisfactorio en todos los navegadores que se han probado, y poco más se puede decir.

5.5 HP Tablet PC TouchSmart TX2-1350

Finalmente, este viejo portátil convertible de hace 5 años, se ha usado en gran medida para el desarrollo y testeo de los eventos táctiles. El rendimiento observado es similar al caso de la maquina con la CPU Core 2 Duo con algunas sorpresas.

Firefox tiene problemas para mover con suavidad decente los ejemplos con mayor cantidad de entradas en GNU/Linux. Aun así, esta dentro de lo aceptable. Chrome y los navegadores basados en webkit, por supuesto mueven el *widget* con total suavidad. Sin embargo, aquí hemos visto que Internet Explorer 11 tiene problemas para mover con suavidad.

6. Conclusiones y desarrollo futuro

En primer lugar, debe destacarse que se han cumplido los objetivos planteados en el inicio del proyecto, así como se han satisfecho los requerimientos del usuario. Desde un punto de vista personal, sería deseable mejorar la suavidad percibida del *widget* en dispositivos limitados como móviles de gama media-baja, pero el nivel conseguido es el adecuado a las necesidades del cliente. Además, el *widget* se puede usar solo o con el wraper, y es relativamente fácil de usar.

Hay varios aspectos que se pueden mejorar, empezando por hacer cierta limpieza de código redundante que aun hay de la fase del prototipo inicial, y acabando por nueva funcionalidad. Aparte, que cualquier mejora de rendimiento sería de agradecer.

Las funcionalidades extra que se podrían incorporar en un trabajo futuro son :

- Modo vertical para Tardis TimeLine : Un modo en que se muestre en una orientación vertical.
- Modo de tiempo flotante : Un modo en que aparte del set de datos suministrado (o que no haya set de datos), se genere un evento cada vez que se desplaza la ventana de tiempo y que el tiempo “seleccionado” siempre sea el centro de la ventana. Un uso de este método, podría ser la actualización dinámica de mapa históricos de imperios y naciones a lo largo del

tiempo.

- Introducción de test de unidad en el entorno de desarrollo, para evitar regresiones de funcionalidad.
- Mayor control del layout generado con jQuery UI Layout. Con dicho mayor control, se podría indicar en cual panel se desea que este el TimeLine o el mapa.
- Opción para que no se use el mapa o el panel en Tardis.js. En ciertos sets de datos, la información geografía no es lo suficientemente interesante o dispersa como para que sea realmente útil el mapa asociado, por ejemplo en el ejemplo de "JFK", el mapa es un tanto superfluo ya que la mayoría de los eventos ocurren en la misma ubicación genérica.
- Mejorar el chequeo de URLs para la carga dinámica. Actualmente es excesivamente laxo, y fácilmente puede dar falsos positivos.
- Re-escribir el código para que reutilice elementos del árbol DOM, en vez de generar todas las banderas. Ya que **vis.js** usa este método y parece funcionar satisfactoriamente, permitiría ahorrar memoria y mejorar el comportamiento del *widget*.
- Pulir algunos defectos menores encontrados en Internet Explorer.
- Mejorar el minimapa, añadiendo alguna indicación visual de los eventos que hay, ya sea como alguna clase de marca sutil similar a los minimapas de SIMILE Timeline.
- Mejorar la carga asíncrona de los contenidos para que descargue las entradas más viejas y permita ahorrar memoria RAM.

7. Agradecimientos

Debo de agradecer enormemente a la empresa y compañeros de Digibís S.L y la Fundación Ignacio Larramendi, sin los cuales este trabajo no se hubiera podido realizar. También agradezco a mis compañeros del proyecto Trillek, que en alguna ocasión actuaron de beta testers.

Agradecimientos en especial para Andrés Viedma, con el cual he debatido bastante ideas de diseño y funcionalidad, además de cómo solventar problemas que han ido surgiendo en el desarrollo del proyecto. También ha sido el encargado de integrar este trabajo en el producto Digimus de la empresa Digibís. También agradecimientos a Jesús Muriel de Digibis, el cual fue quien tuvo la idea inicial, de la cual desarrolle el proyecto. Finalmente, agradecimientos a la labor de supervisión del tutor de este trabajo, Alvaro Ortigosa.

8. Anexo Técnico

8.1 Opciones de tardis-tl.js

Este anexo refleja el contenido de la *wiki* a fecha del 06-07-2014

Tardis TimeLine aka Tardis-TL

V 0.2

jQuery Plugin that implements a widget to display and interact with time events in a*time-line* display

Requisites and recommendations

Actually needs: * jQuery 1.11 (Obvisuly). Probably could work with older versions, but not tested.

- jQuery UI 1.10 Slider widget.
- jQuery.hammer.js is optional but recommend.

Also, we recommend :

- moment.js is recommended to build powerful **formatter** and **parser** functions.

Usage

`$('...').TardisTL(data)` : Initializes Tardis TimeLine
`$('...').TardisTL(data, options)` : Initializes Tardis TimeLine and pass configuration options

Data

Data is an array of objects, were each entry in the list must have at least this entries:

Entry	Description
start	Start/Born year. Will be parsed by **parser** function
end	End/Dead year. (Optional)
title	Title/Name of the entry. Tardis TimeLine unwraps any HTML link detected on it.

Options

Options are passed to Tardis TimeLine using a JavaScript dictionary. The entries of these dictionary could be :

start

Start of the range of visible dates. If is missing, Tardis TimeLine will calculate it. It will be parse by **parser**function.

end

End of the range of the visible dates. If is missing, Tardis TimeLine will calculate it. It will be parse by**parser** function.

min_zoom

Number of years to show when zoom out. By default is 'auto', where will try to calculate an optimal value.

max_zoom

Number of years to show when zoom in. By default is 'auto', where will get 1/30 of *min_zoom*.

initial_zoom

Initial zoom level. By default is 'auto', where will use the value of *min_zoom*.

max_rows

Max number of rows used to show the flags markers. By default is 500.

row_sep

Separation in pixels (Number) between each row. By default is 5 pixels.

fixed_height

If is False, TardisTL will resize itself to accommodate the necessary rows to display the flags. By default is False.

minimap

If is true, will show a *minimap* that the user can slide, to displace the visible time window. By default is True.

parser

Function that parses the start/end dates of each entry and start/end range of visible dates to a number. Tardis TimeLine abstracts itself of time unit being used. By default it pass the value.

formatter

Function returns the String representation of the date/time value.

Formatter could accept some parameters listed here in order:

- **value** : Abstract time unit used by Tardis TimeLine to be formatted.
- **where** : String that indicates where is called the formatter function, could be "cursor", "T0", T1",..., "M".
- **orig** : Raw original start date (only when where is "cursor" and the view is over a selected item). By default is :

```
function (value /*, where, original*/ ) { // Formatting function for user presentation of ti
  if (value < 0) {
    return String(-value) + ' b.C.';
  } else {
    return String(value);
  }
}
```

marks

Defines the Timeline "time" marks. It's array that in each entry have a dictionary with the fields:

- **start** : Where the marks begin. If it's omitted, Tardis-TL will try to guess the appropriate value using this formula : Start = floor(start_of_dataset / step) * step . With this, try to round dates so for example setting a step to generate marks every 100 years, will be aligned to 0, 100, 200, 300, etc...
- **end** : Where the marks ends. By default gets the end date.
- **threshold** : If Pixels Per Time Unit is below this value, hides this mark set.
- **step** : Could be a value or a function that return the new value from the previous value. In both cases, values are in abstract time units, and is used to generate every mark of the set.
- **dclass** : String with CSS Class assigned to the mark. Useful to apply custom CSS style to every mark set. This value is passed to **formatter** function as **where** value.

minimap_marks

Defines the *Minimap* "time" marks. It's a dictionary with the fields:

- **start** : Where the marks begin. If it's omitted, Tardis-TL will try to guess the appropriate value using this formula : Start = floor(start_of_dataset / step) * step . With this, try to round dates so for example setting a step to generate marks every 100 years, will be aligned to 0, 100, 200, 300, etc...
- **end** : Where the marks ends. By default gets the end date.
- **step** : Could be a value or a function that return the new value from the previous value. In both cases, values are in abstract time units, and is used to generate every mark of the set.
- **dclass** : String with CSS Class assigned to the mark. Useful to apply custom CSS style to every mark set. This value is passed to **formatter** function as **where** value.

Methods

Tardis TimeLine exposes some methods to interact with it in JavaScript code :

Method	Description
`$('...').TardisTL(number)`	Go To data[number] entry.
`$('...').TardisTL('setEntry', number)`	Go To data[number] entry.
`$('...').TardisTL('setView', number)`	Move the view to the year number.
`$('...').TardisTL('setZoom', number)`	Sets zoom factor.
`$('...').TardisTL('redraw')`	Forces to update screen state of the widget.
`$('...').TardisTL('index')`	Returns actual selected index.

Events

Tardis TimeLine have some events to execute interactive code in respond to the user actions on the widget :

Event	Description
tardistl.flag	Launched BEFORE changing the selected time event.
tardistl.flagn	Launched AFTER changing the selected time event.
tardistl.resize	Launched AFTER TardisTL resizes itself when fixed_height option is false.

Events *tardistl.flag* and *tardistl.flag*, have a parameter that is a dictionary with the entries **newindex** and**oldindex**. **oldindex** stores the the index of the previous selected time event, and **newindex** have the index of the new selected time event.

8.2 Opciones de tardis.js

Este anexo refleja el contenido de la *wiki* a fecha del 06-07-2014

Tardis

V 0.2

jQuery plugin that implements a meta-widget to display generic TimeLines with multimedia data.

Integrates **Tardis TimeLine** with a Leaflet map and self implemented carousel, to display a generic time line of events with multimedia data. It can use his own layout scheme using **jQuery UI Layout** to emplace the widgets in resizeable panes. If not detects jQuery UI layout being loaded, then will simple put each widget in different divs. Also, allows to assign independent jQuery objects that will be used to each widget, allowing to use any desired framework to layout each widget.

Requisites and recommendations

Actually needs:

- jQuery 1.11 (Obvisuly). Probably could work with older versions, but not tested.
- LeafLet 0.7.2
- jQuery UI 1.10 slider widget
- jQuery UI Layout 1.3.0 but could be used without it if **ui_layout** is set to 'none' or 'external'.
- jQuery UI 1.10 core and ui.draggable.js if is being used with jQuery UI Layout

Also, we recomend :

- jQuery.hammer.js is optional but recommend.
- jquery.ui.touch-punch is optional but recomended if you use jQuery UI Layout with tactile interactions.
- moment.js is recommended to build powerful **formatter** and **parser** functions.

Usage

`$('...').Tardis(data)` : Initializes Tardis
`$('...').Tardis(data, options)` : Initializes Tardis and pass configuration options

Data

Data could be an string with a URL to download JSON data or an array. If Data is a URL to do a *async*download of data, is expected that the data consists in an array of objects. If Data is an array, is expected that the data consists in an array of objects.

Each object must have this entries:

Entry	Description
start	Start/Born year.
end	End/Dead year. (Optional)

title	Title/Name of the entry. Tardis TimeLine unwraps any HTML link detected on it.
location	Array with Lat & Log coordinates. If is a empty array, will be interpreted like a time event without location. (Optional)
place	Name of the location. (Optional)
map_marker_title	Text to show in the map marker popup. If is missing, fall-backs to title. (Optional)
description	HTML body of the entry. Could contain any HTML data that could be pu inside a . Also, could be a URL/URI to a JSON source that will loaded.

Options

Options are passed to Tardis using a JavaScript dictionary. The entries of these dictionary could be :

tardistl

Sub-dictionary witch options to **Tardis TimeLine**. See TardisTimeLine

leaflet

Sub-dictionary with the options for **Leaflet**. Contains some default values :

tilemap_url

Format URL that needs leaflet to use a tesseras source. By default points to OpenStreetMap.

attributuon

Attribution text about the tesseras source.

min_zoom

Minimal zoom factor, that is set to 1.

max_zoom

Max zoom factor, that is set to 18.

ui_layout

String that chooses if it will generate a layout or not. Could be :

1. 'none' : simply will generate divs for each widget and nothing more.
2. 'jq_ui_layout' : Uses **JQuery UI Layout** to create panels for each widget (Default value)
3. 'extern' : Uses an external create layout. It's necesary set XX_container options with the apropiated values to use this layout option.

jq_ui_options

Options to send to **jQuery UI Layout** (See jQuery UI Layout documentation), if is used 'jq_ui_layout' layout. By default sets, a few values in function of if TardisTL 'fixed_height' option is false, and if the browser is running in a phone or tablet.

tardistl_container

jQuery object were **Tardis TimeLine** will be generated, if is used 'extern' layout.

map_container

jQuery object were **Leaflet** will be generated, if is used 'extern' layout.

carousel_container

jQuery object were own carousel will be generated, if is used 'extern' layout.

formatter

Function returns the String representation of the date/time value. If it's set, overwrites the **formatter** that uses the internal TardisTL.

Formatter could accept some parameters listed here in order:

- **value** : Abstract time unit used by Tardis TimeLine to be formatted.
- **where** : String that indicates where is called the formatter function, could be "cursor", "T0", T1",..., "M".
- **orig** : Raw original start date (only when where is "cursor" and the view is over a selected item). By default uses the same default function that **TardisTL**.

formater_int

Function that returns the String representation of a interval date/time value. By default apply **formatter**to both values and puts a " - " between both values.

dynamic_load

Flag that enables dynamic load of data. If this mode is active, Tardis will analyse if description field is a URL/URI, and if is it, will try to do a AJAX petition of these URL/URI when the user selects these time event.

It expects to get a JSON data with a "description" field that will be used to display the multimedia data of the entry.

fallback_location

If any time event not have a location field or have a invalid location field, will use this value. If this values is a empty list (i.e. a []), then will not use a entry with a invalid location field. By default is [], so Tardis will interpret it as a "without location" time event.

fallback_initial_location

If the initial time event is a "without location" time event, this coords will be used as initial location by the map. By default is [0, 0] that is a place in the Atlantic ocean, near Africa.

no_location_str

Message to show when a entry not have a valid or know location.

shows_loading

Shows a generic loading image over the whole page, while Tardis is loading and generating all stuff. By default is true.

If **tardistl_container** , **map_container** and **carousel_container** are at same time assigned to jQuery objects and **ui_layout** is set to 'external', they will be used and Tardis will not generate his own layout. This is useful if you desired to use Bootstrap or any other similar framework. You can see a example of this in **23-F TimeLine demo**.

Methods

Tardis exposes some methods to interact with it in JavaScript code :

Method	Description
`$('...').Tardis('getJQMap')`	Returns jQuery object that contains the map.
`$('...').Tardis('getLeafletMap')`	Returns LeafLet object.
`$('...').Tardis('getTardisTL')`	Returns jQuery object of TardisTL (so can be used to add events or call methods of Tardis TimeLine).
`$('...').Tardis('getPanel')`	Returns jQuery object that contains carousel.

Referencias

1: Ramon Manuel Gonzalvo Mourelo , Eje cronológico de Historia de España (Siglos XIX y XX) , http://entendiendolahistoria.blogspot.com.es/2011/04/eje-cronologico-de-historia-de-espana.html
2: Wikipedia, Definiciñon de polígrafo, http://es.wikipedia.org/wiki/Pol%C3%ADgrafo_(autor)
3: Northwestern University Knight Lab, timeline.js, http://timeline.knightlab.com/
4: Massachusetts Institute of Technology, SIMILE Timeline, http://www.simile-widgets.org/timeline/
5: Almende B.V. , vis.js A visual interaction system, http://visjs.org/
6: Y Combinator; BKCandace, Hacker News, https://hn.algolia.com/?q=vis.js#!/story/forever/0/vis.js
7: , Bug de Calendar en Java, http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4639407
8: Jorik Tangelder at Eight Media, Hammer.js, http://eightmedia.github.io/hammer.js/
9: Mathias Bynens, In search of the perfect URL validation regex, http://mathiasbynens.be/demo/url-regex